Naval Research Laboratory

Washington, DC 20375-5320

NRL/MR/5542--05-8903

(3)

Security Ontology for Annotating Resources

ANYA KIM
JIM LUO
MYONG KANG

Center for High Assurance Computer Systems
Information Technology Division

August 31, 2005

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
31-08-2005	Memorandum Report	

4. TITLE AND SUBTITLE

Security Ontology for Annotating Resources

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Anya Kim, Jim Luo, and Myong Kang

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
55-8089-G5

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory, Code 5542
4555 Overlook Avenue, SW
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/MR/5542--05-8903

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
Ralph Wachter, Code 311
800 North Quincy Street
Arlington, VA 22217

10. SPONSOR / MONITOR'S ACRONYM(S)

11. SPONSOR / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Annotation with security-related metadata enables discovery of resources that meet security requirements. This paper presents the NRL Security Ontology, which complements existing ontologies in other domains that focus on annotation of functional aspects of resources. Types of security information that could be described include mechanisms, protocols, objectives, algorithms, and credentials in various levels of detail and specificity. The NRL Ontology is more comprehensive and better organized than existing security ontologies. It is capable of representing more types of security statements and can be applied to any electronic resource. The class hierarchy of the ontology makes it both easy to use and intuitive to extend. We applied this ontology to a Service Oriented Architecture to annotate security aspects of Web service descriptions and queries. A refined matching algorithm was developed to perform requirement-capability matchmaking that takes into account not only the ontology concepts, but also the properties of the concepts.

15. SUBJECT TERMS

Security; Ontology; Semantic Web

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Anya Kim
Unclassified	Unclassified	Unclassified	UL	54	19b. TELEPHONE NUMBER *(include area code)* (202) 767-6698

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

Table of Contents

Security Ontology for Annotating Resources

Anya Kim, Jim Luo, and Myong Kang

Center for High Assurance Computer Systems
Naval Research Laboratory
Washington, DC 20375
{kim, luo, mkang}@itd.nrl.navy.mil

Abstract

Annotation with security-related metadata enables discovery of resources that meet security requirements. This paper presents the NRL Security Ontology, which complements existing ontologies in other domains that focus on annotation of functional aspects of resources. Types of security information that could be described include mechanisms, protocols, objectives, algorithms, and credentials in various levels of detail and specificity. The NRL Security Ontology is more comprehensive and better organized than existing security ontologies. It is capable of representing more types of security statements and can be applied to any electronic resource. The class hierarchy of the ontology makes it both easy to use and intuitive to extend. We applied this ontology to a Service Oriented Architecture to annotate security aspects of Web service descriptions and queries. A refined matching algorithm was developed to perform requirement-capability matchmaking that takes into account not only the ontology concepts, but also the properties of the concepts.

1. Introduction

In today's network-centric military environment, automatic discovery of resources and the ability to share information and services across different domains are important capabilities [7]. The first step in providing these capabilities is to markup these resources with various metadata in a well-understood and consistent manner. Such annotation will enable resources to be machine-readable and machine-understandable.

Using metadata to find distributed resources that meet one's functional requirements is only the first step. Resource requestors may have additional requirements such as security, survivability, or quality of service (QoS) specifications. For example, they may require resources to possess a certain military classification level, to originate from trusted sources, or to be handled according to a specified privacy policy. Therefore, resources need to be sufficiently annotated with security-related metadata so that they can be correctly discovered, compared, and invoked according to security as well as functional requirements of the requestor.

In this paper, we introduce a set of security-related ontologies collectively referred to as the NRL Security Ontology. The NRL Security Ontology provides the ability for precisely

Manuscript approved June 13, 2005.

describing security concepts at various levels of detail. This ontology complements existing ontologies that mainly focus on functional aspects of capability, content, and parameters. Marking up security aspects of resources is a crucial step toward deploying a secure Service Oriented Architecture (SOA) system.

Other groups have recognized the need for security annotation of services and proposed a set of security-related ontologies [2-4]. However, these ontologies lack the ability to express certain security concepts, contain unnecessary concepts, and are organized in a non-intuitive way. The NRL Security Ontology was created to address these limitations. We expect this work to serve as a catalyst in the development of standardized security-related ontologies with contributions from both the security community and the semantic Web community.

The rest of the paper is organized as follows. Section 2 examines previous work in security ontology and discusses the need for improvements. Section 3 presents the NRL security ontology, including design objectives, domain and scope, and detailed descriptions. Section 4 gives examples of how to use these ontologies to annotate and query for resources particularly in a Web service context. It also discusses our algorithm for matchmaking between queries and resource descriptions. Section 5 presents future work and our conclusion.

2. Existing Security-related Ontologies

Realization of the need for security ontologies is not new. Denker et al. have created several ontologies for specifying security-related information in Web services [2] using Daml+OIL [5] and later OWL [6]. We refer to this set of ontologies as the DAML Security Ontology for the rest of the paper. The authors state that the goal of these ontologies is to enable high-level markup of Web resources, services, and agents, while providing a layer of abstraction on top of various Web service security standards such as XML-Enc [7], XML-Dsig [8], and SAML (Security Assertion Markup Language) [9].

Of the set of ontologies that make up the DAML security ontology, the two main ontologies are the Security Mechanisms ontology and the Credential ontology. They describe security mechanisms and authentication credentials respectively. While we realize that these ontologies are works-in-progress, we found two issues with them. First, they are not intuitive to understand especially in terms of the organization of subclass relationships. Second, they cannot express all the security information that we want to describe.

The intuitiveness issue is particularly true for the main Security Mechanisms ontology. Figure 1 depicts this ontology in a simplified form where circles denote classes, solid lines represent instances of the classes and dotted lines represent properties[1]. The top class in this ontology is 'SecurityMechanism' with subclasses of 'SecurityNotation', 'Signature', 'Protocol', 'KeyFormat', 'Encryption', and 'Syntax'. Making these unrelated concepts sibling classes does not make sense from either a security perspective or an ontology perspective. Furthermore, some instances are not properly assigned to the correct subclass. For example, Kerberos and SSH are both declared as instances of 'KeyProtocol', however these are not key protocols. Additionally, all properties are defined for the top class. However, those properties

[1] Some complex concepts they use such as restriction classes are not depicted here.

do not apply to most of the subclasses. For example, no instance under the 'Syntax' subclass would have a need for the *relSecNotation* (Relative Security Notation), *enc* (Encryption), *sig* (Signature), or *reqCredential* (Required Credential) properties, yet they are all inherited because these properties are defined at the top class.

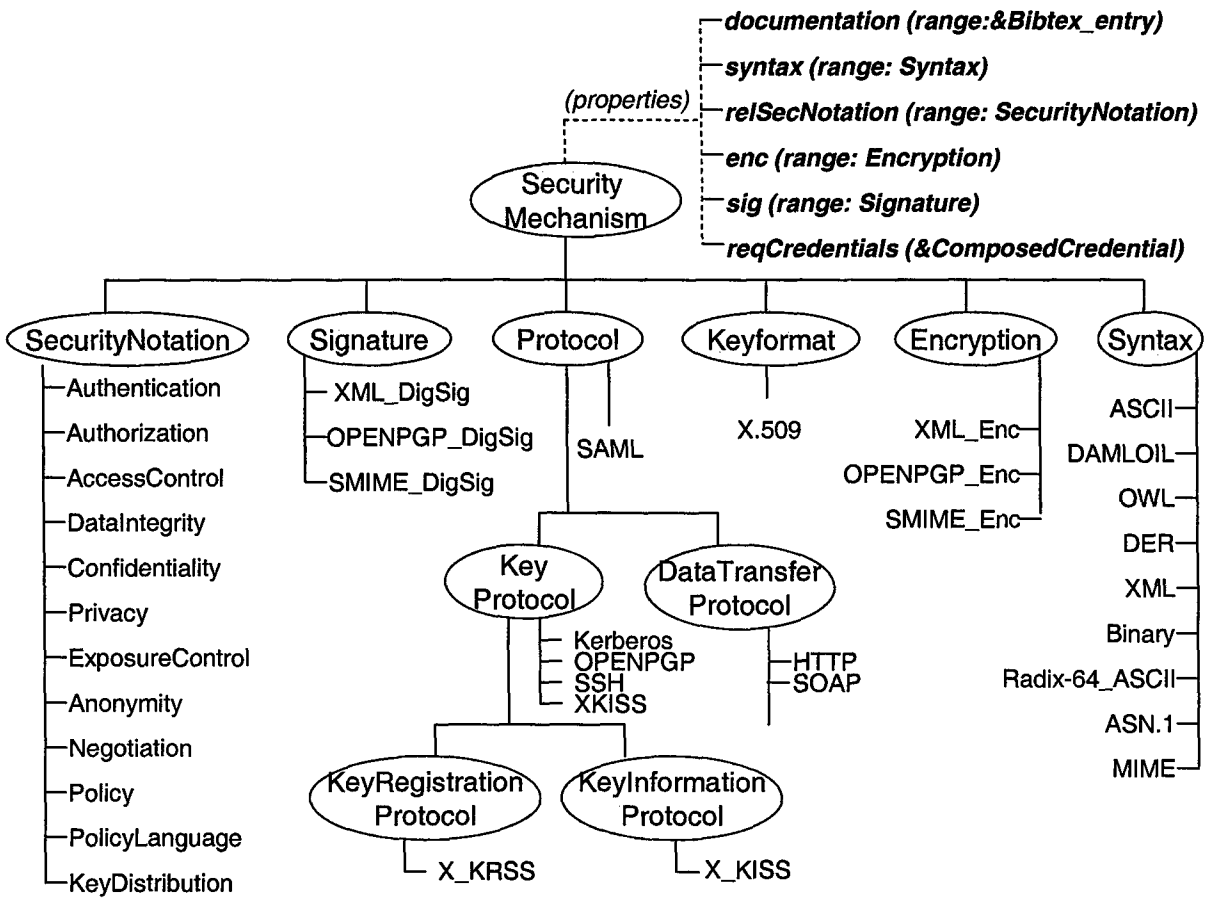


Figure 1: Simplified DAML Security Mechanisms Ontology

The second issue we mentioned is the lack of expressiveness. The DAML security ontology includes many classes and instances that are not directly relevant for security annotation while lacking others that are necessary. For example, syntax and data transfer protocols are useful concepts in another domain, but are not particularly relevant for describing security-related information. Furthermore, the only encryption instances defined in the ontology are S/MIME, OpenPGP, and XML encryption. We do realize that more instances could be added as the need arises. However, the organization of the class hierarchy should be well developed. For example, there should be classes to represent military as well as commercial security devices and security policies. Currently, there is no appropriate place in the DAML Security Ontology to create a firewall or military security policy instance. There is also a lack of appropriately placed properties that could allow for more detailed refinement of security concepts. For example, it would be useful to define the algorithms supported by a protocol, or the certification status of a mechanism.

Although the authors of the DAML Security Ontology did a great job in recognizing the need for security ontologies and beginning work in security ontologies, we feel that there is still room for improvement. The next section describes the NRL Security Ontology in detail.

3. NRL Security Ontology

The DAML Security Ontology focuses on annotation of Web services rather than resources in general. This is evident not only from their documentation [2], but also by examining the types of classes and instances in the ontology. We want ontologies that can be used to annotate generic resources from simple documents to interactive services with security-related metadata. We also want to improve upon the limitations of the DAML Security Ontology outlined in the previous section. The NRL Security Ontology was designed with the following objectives in mind:

1. Describe security related information applicable to all types of resources
2. Provide the ability to annotate security related information in various levels of detail for various environments (both commercial and military)
3. Create ontologies that are easy to extend and provide reusability
4. Facilitate mapping of higher level (mission-level) security requirements to lower-level (resource level) capabilities

3.1. Domain and Scope of the Ontology

When creating an ontology, one of the most important factors is the domain and scope in which it will be used [10]. While our objectives outlined above are a good starting point, in order to create ontologies that will be truly useful, we need to understand the types of questions that the ontology will be expected to answer.

These ontologies will be used by both the resource provider and the requestor to express their security requirements and capabilities. We must consider the various ways that the same statement can be expressed. Furthermore, we need to consider statements that are unlikely in order to limit the scope of the ontology. Statements that are either too broad or too specific are unlikely to be used and provide no useful information.

Noy et al. [10] state that one of the best ways to determine the scope of the ontology is to list a set of *competency questions* that can be answered using the ontology. For our purposes we did the same by composing a list of security requirements and capabilities for both the resource requestor and the provider. From the requestor's perspective, security requirements can be stated in terms of specific mechanisms or in terms of abstract security objectives. From the resource provider's perspective, security requirements are similar to the notion of policy and can express concepts such as authentication and access control. The provider's capabilities include protocols and mechanisms that the provider possesses and security policies it adheres to. The actual list of the requirements and capabilities statements we created can be found in Appendix B.

3.2. Organizational Structure of NRL Security Ontology

We chose OWL to create our ontologies because it provides a rich vocabulary for describing classes and properties [6, 11]. It is widely used in many communities that have begun to develop ontologies of their own knowledge domains [12].

There are seven separate ontologies that make up the NRL Security Ontology:

1. Main Security ontology: an ontology to describe security concepts
2. Credentials ontology: an ontology to specify authentication credentials
3. Security Algorithms ontology: an ontology to describe various security algorithms
4. Security Assurance ontology: an ontology to specify different assurance standards
5. Service Security ontology: an ontology to facilitate security annotation of semantic Web services
6. Agent Security ontology: an ontology to enable querying of security information
7. Information Object ontology: an ontology to describe security of input and output parameters of Web services

The Service Security, Agent Security, and Information Object ontologies are modifications and extensions of some existing DAML Security ontologies while the others are new. The Credentials, Security Algorithms, and Security Assurance ontologies provide values for properties defined for concepts in the Main Security ontology. They enable those concepts to be described in more detail with respect to types of credentials used, supported algorithms, and associated levels of assurance. The Service Security ontology provides the means to use security concepts from the Main Security ontology in the Web services framework. The Agent Service ontology enables creation of security-related queries using security concepts from the Main Security ontology. The Information Object ontology allows for annotation of Web service inputs and outputs using the Security Algorithm ontology. The relationship among these ontologies is represented in Figure 2. The ontology depicted in gray represents OWL-S, a set of core ontologies used to describe Web services.

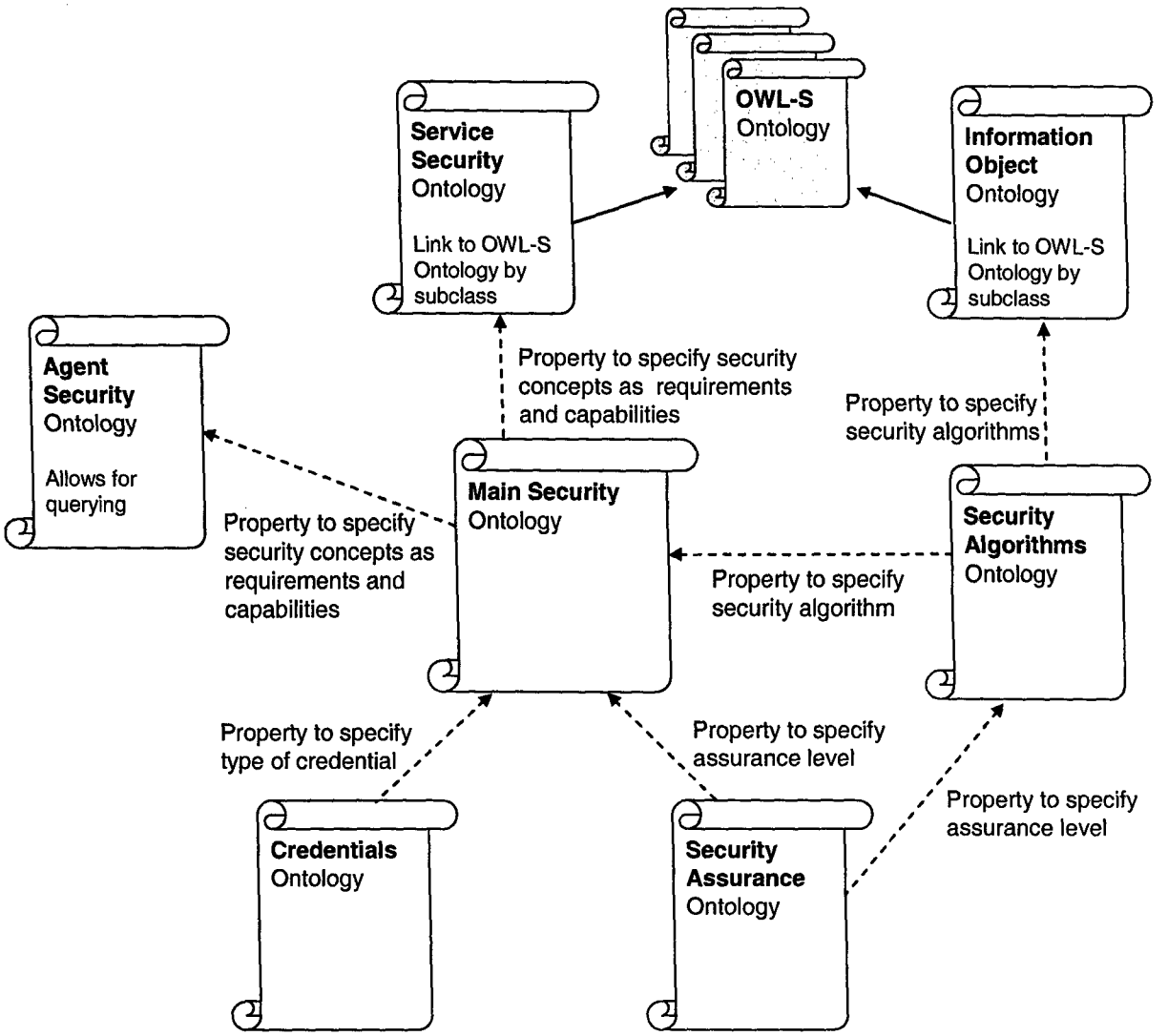


Figure 2. Graphical Representation of Security-Related Ontologies and Their Relationships

Next, we present a brief explanation of classes, properties and relationships in each ontology. A complete graphical depiction of these ontologies can be found in Appendix C and the OWL files for each ontology in Appendix D.

Main Security Ontology (securityMain.owl)

The core ontology in the NRL security ontology set is the Main Security ontology (Figure 3). It imports the Credentials ontology, Security Algorithms ontology, and Security Assurance ontology as object properties. The top class, 'SecurityConcept' possesses three subclasses: 'SecurityProtocol', 'SecurityMechanism' and 'SecurityPolicy'.

While some may argue that the distinction between security protocols and security mechanisms is blurred, we define security protocols as an agreed upon series of steps to accomplish a task while security mechanisms are implementations of protocols [13]. We

specifically differentiate them here to provide the ability to describe security in both manners. Security policies are the set of rules that regulate how information is protected and secured .

The Main Security ontology also has a separate class called 'SecurityObjective' that enables users to specify security objectives for the 'SecurityConcept' class using the *supportsSecurityObjective* property. For example, IPSec is declared to have Confidentiality, MessageAuthentication, and TrafficHiding as its *supportsSecurityObjective* property values. Security objectives also enable users to search for protocols, mechanisms, or policies based on the security objective they require. For example, users can query, "find all instances that provide confidentiality" and receive a list of all the security concepts that have a value of Confidentiality in their *supportsSecurityObjective* property.

Another way we can use 'SecurityObjective' is to map high-level mission requirements to low-level service requirements. For instance, assume that a security requirement is specified at the mission level such that Mission 1 and Mission 2 must have separation between them. At this level, the mission planner can use the ontology to specify the security objective of Separation. The mission designer can then search for instances in the 'SecurityConcept' class provide Separation. In this case, the only one that does is VPN, so he can select VPN as a security requirement at the service level.

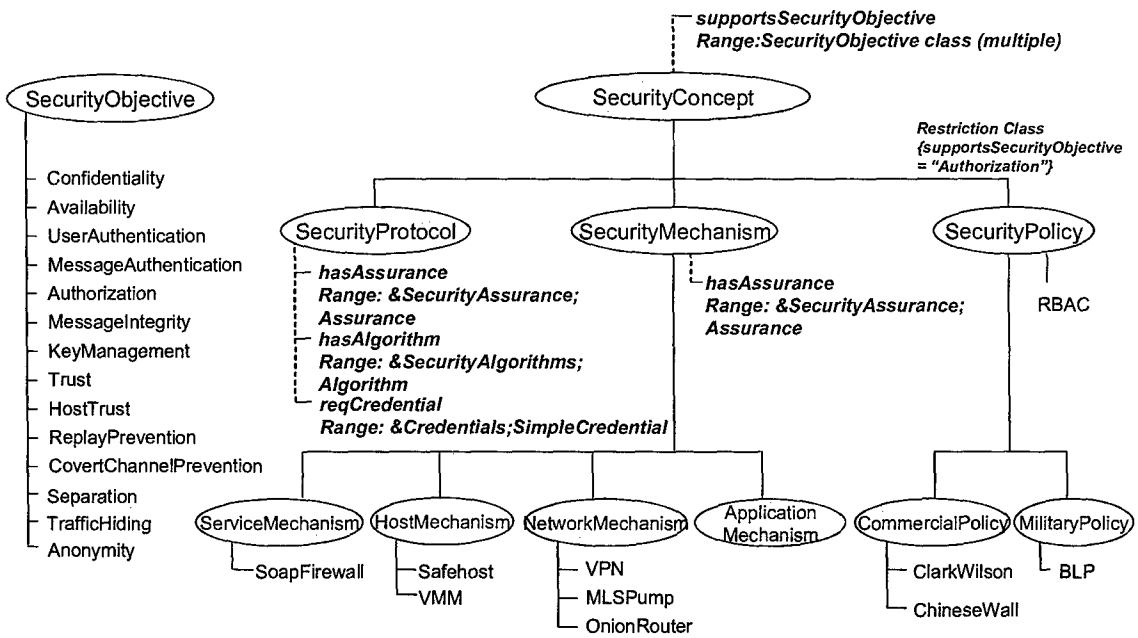


Figure 3: A Part of the Main Security Ontology

Credentials Ontology (credentials.owl)

Authentication is one of the most fundamental security requirements in a networked environment. The Credentials ontology allows for specification of credentials used for authentication purposes. Concepts in the Security Main ontology can refer to a specific

credential through their *reqCredential* property. While we adopted some of the notations in the DAML Credential ontology, we found that the organization of various credentials in the DAML Credential ontology was not intuitive and some of the classes were unnecessary. We improved upon their Credential ontology by reorganizing classes to be more intuitive, including more properties and adding more classes to define additional types of credentials. Our Credentials ontology categorizes credentials into physical token, electronic token, and biometric token.

Under the 'PhysicalToken' class, we kept many of the classes from the DAML Credential ontology under their 'IDCard' class. In addition, we created a class to describe military IDs and an instance to represent CAC (Common Access Card) cards used in the military. The ontology can be extended to add properties such as issuing agency, expiration date, issue date, etc. Under the 'ElectronicToken' class, we provide subclasses that enable authentication based on host address, certificates, passwords, and cryptographic keys to name a few. Additional properties were added to describe certificates including the issuer, version and serial number under the Certificate class. In order to support role-based (RBAC) certificates [14], an 'RBACCertificate' class was created as a subclass of the Certificate class with a *role* property. The 'BiometricToken' class represents credentials that pertain to human traits. For now, only 'Voice' and 'Fingerprint' subclasses are defined here.

In addition to the three categories of simple credentials, the 'MultifactorCredential" class can be used to describe composed credentials made up of two or more individual credentials. For example, it can describe requirements where both a smart card as well as a password is needed.

Security Algorithms Ontology (securityAlgorithms.owl)

The Security Algorithms ontology was created to enable description of various security algorithms (Figure 4). It can be used to describe not only a list of algorithms that a given security protocol can support, but also to specify what algorithm was used to encrypt or sign given data objects as inputs or outputs of a Web service. It classifies these algorithms broadly into encryption, signature, key exchange, and checksum algorithms. It provides a means to describe whether or not a given algorithm is a government standard through the *isNISTStandard* property. It can also describe whether an encryption algorithm is military Type1, Type 2, Type 3, or Type 4 [15] through the *hasNSALevel* property. Additional properties enable description of key lengths and modes of operation.

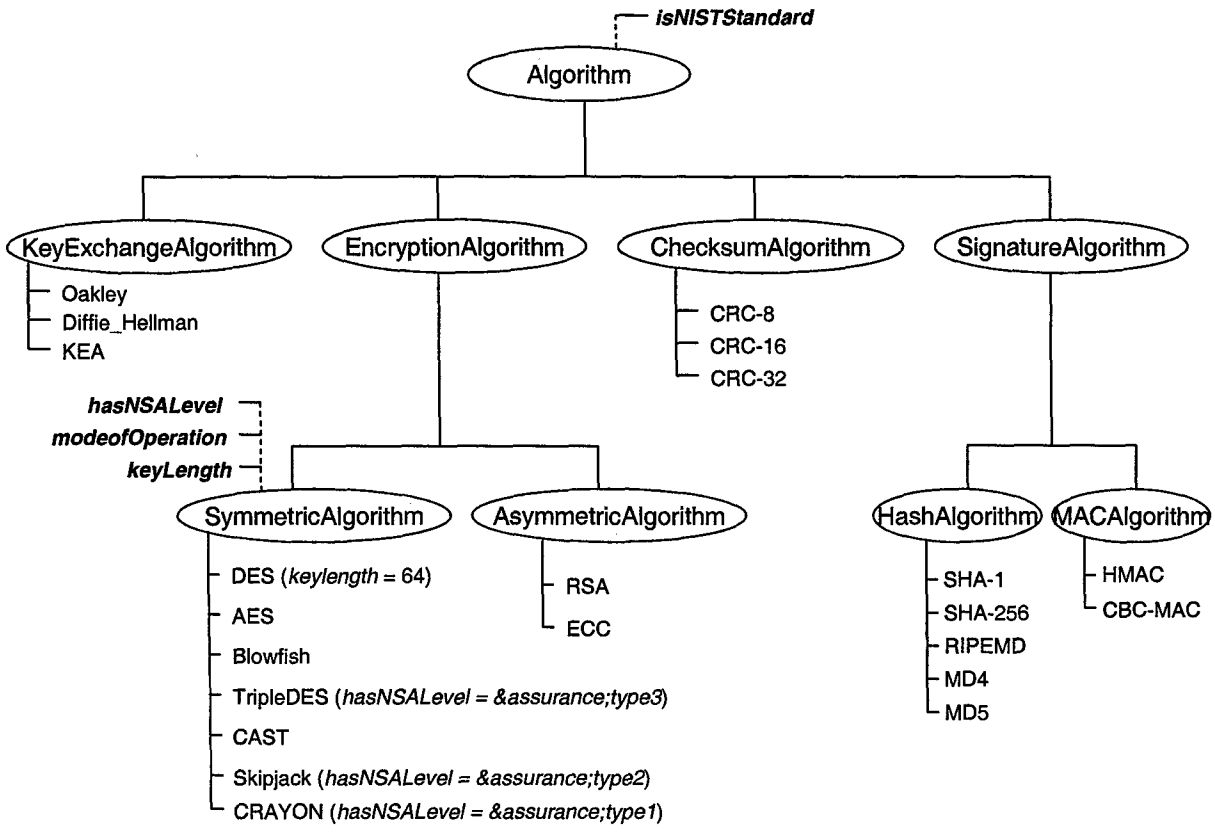


Figure 4: Security Algorithms Ontology

Security Assurance Ontology (securityAssurance.owl)

The Security Assurance ontology provides a way to describe standardized assurance methods for security protocols, mechanisms, and algorithms. They can be described in terms of their assurance level using the *hasAssurance* property from the Main Security ontology. The 'Assurance' class is classified according to different assurance methods: 'Standard', 'Accreditation', 'Evaluation', and 'Certification'. This ontology is the least compete of all our ontologies. However, we have added classes to describe the Common Criteria and TCSEC evaluations, and the FIPS and NSA standards [15].

Service Security and Agent Security Ontologies (serviceSecurity.owl and agentSecurity.owl)

OWL-S [16] is an OWL-based semantic markup description language that provides a core set of constructs for describing Web services specifically. It provides a set of ontologies called Profile, Process, and Grounding to describe Web services. The Profile describes services in terms of what the service does, the Process describes how to use it, and the Grounding specifies how to interact with it.

In order for the NRL Security Ontology to be used in the Web service context, a link must be made to the OWL-S ontologies. The Service Security ontology was developed for such a

purpose. In the Service Security Ontology, 'SecurityConcept' and 'SecurityObjective' from the Main Security ontology are defined to be subclasses of the 'ServiceParameter' class in the OWL-S Profile ontology (Figure 5). The OWL-S Profile also contains a *serviceParameter* property that can have ServiceParameter as its value[2]. Declaring two subproperties of the *serviceParameter* property, *securityRequirement* and *securityCapability* enables the OWL-S Profile to include security requirements and security capabilities in its service description. Furthermore, we defined the range for these subproperties as either the 'SecurityConcept' or 'SecurityObjective' classes. This allows security requirements and capabilities to be stated in terms of either a particular security objective, or a specific security mechanism.

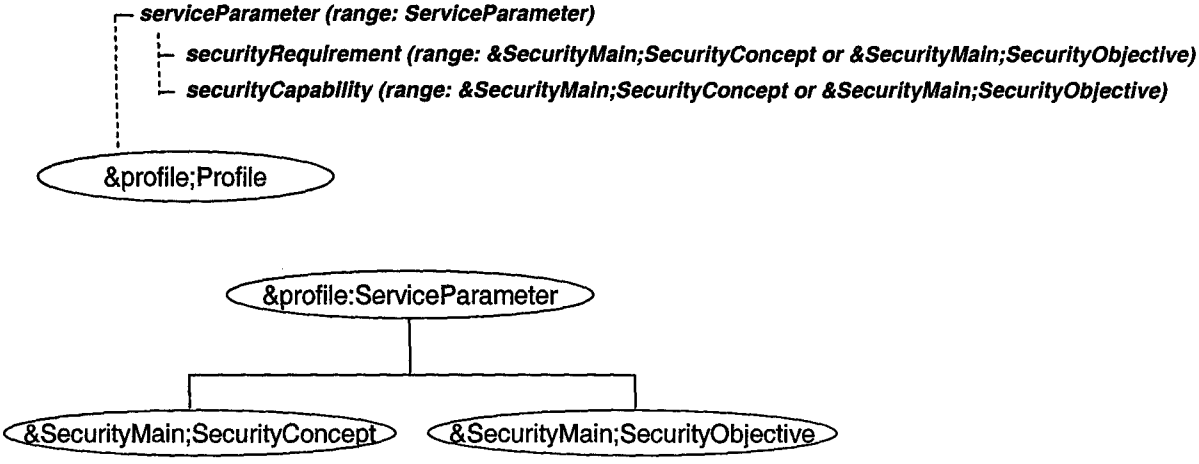


Figure 5: Service Security Ontology

The Agent Security ontology allows for querying of resources, in particular Web services with requestor requirements and capabilities. It defines an 'Agent' class to represent the service requestor with the properties *securityCapability* and *securityRequirement* that can hold values from the 'SecurityConcept' and 'SecurityObjective' classes.

Information Object Ontology (InfObj.owl)

The Information Object ontology is based on a DAML ontology created to capture encrypted or signed input/output data of Web services. It has an 'InfObj' class and two subclasses, 'EncInfObj' (Encrypted Information Object) and 'SigInfObj' (Signed Information Object). The 'InfObj' class is used as the range for input and output parameters of services described with OWL-S. The ontology has the *cryptoAlgUsed* property to specify the algorithm used to encrypt or sign the object. In the original DAML ontology, the *cryptoAlgUsed* property pointed to a set of algorithms defined within the DAML Information Object ontology. However, we felt that the two concepts of information object and security algorithms were so dissimilar that they did not belong within the same ontology file. Hence, in the NRL Information Object ontology, the *cryptoAlgUsed* property points to classes in the Security Algorithms ontology.

[2] Note that the OWL-S Profile ontology has a property and class of the same name, service parameter. However, the property starts with a lowercase letter, while the class starts with an uppercase letter. Thus, *serviceParameter* refers to a property while ServiceParameter refers to a class.

3.3. Design Objectives Revisited

At the beginning of Section 3 we outlined a set of objectives expected to be achieved by the NRL Security Ontology. This subsection discusses whether those design objectives were met and to what degree.

1. **Describe security related information not only for Web services, but for all types of resources:** The NRL Security Ontology enables us to describe security information of various types of resources. We can describe security protocols that are specific to Web services such as XML-enc and SAML, but also include many protocols and mechanisms such as IPSec, Kerberos and SSH that are generally applied to any resource.
2. **Provide the ability to annotate security related information in various levels of detail for various environments:** The ontology can provide specific details of security mechanisms through properties such as the types of algorithms supported, required key length, types of credentials used, and expiration dates. Classes and instances were created that enable description of resources relevant to a military environment as well as for commercial use.
3. **Create ontologies that are easy to extend and provide reusability:** The ontologies are created with a class hierarchy that makes sense from a security perspective. New instances when necessary can be added to the ontology in an intuitive manner with out having to alter the class hierarchy. Also, we limited the creation of unnecessary ontologies by reusing several of the ones given in the DAML ontology by extending and modifying them. When we had to create new ontologies, we justified our reasons by stating why the new ontology was more suitable.
4. **Facilitate mapping of higher level (mission-level) security requirements to lower-level (resource level) capabilities using the ontology:** resources can be described in terms of either security objectives at the abstract level, or security concepts at the concrete level. A mapping was established so that moving between the two methods of specification is possible.

In the next section, we will provide some examples of how to apply these ontologies to annotate resources with security information.

4. Application of NRL Security Ontology to the Service Oriented Architecture

While the NRL Security Ontology can be used to describe security-related information of resources in general, in this section we discuss how to annotate Web services in a Service Oriented Architecture. In particular, we focus on:

- How to annotate Web service descriptions with security requirements and capabilities
- How to create queries for finding Web services with given security requirements and capabilities
- How to perform matchmaking between queries and service descriptions in the SOA context.

4.1. Reasoning and Matching Algorithm

We have stated that both resource requestors and providers have security requirements and capabilities. Matchmaking looks for a two-way correspondence between these requirements and capabilities. In other words, service requirements are compared to requestor capabilities and service capabilities are compared to requestor requirements. In order for a match of security concepts to occur between a service provider and a service requestor, two conditions should be met. First, the provider's security capabilities should satisfy the requestor's security requirements. Second, the provider's security requirements should be satisfied by the requestor's security capabilities. This implies that the requirements should subsume the capabilities (Table 1).

Table 1. The Matching Relationship between Requestor and Provider Requirements and Capabilities

Requestor		**Provider**
Requirements	$\subseteq$	Capabilities
Capabilities	$\supseteq$	Requirements

Every single requestor requirement must have a corresponding capability on the provider side to satisfy it, and vice versa. Hence the matchmaker must be able to perform two tasks. First, it must be able to determine the level of match for each specific requirement and a specific capability. Second, it must use those levels of match to determine if the set of requirements is matched by the set of capabilities. In other words, the matchmaker must determine the level at which each requirement is matched to a capability, and then the overall level of match between the requester and the provider. This will be explained in detail later in the section.

Several semantic matching algorithms have been proposed [2, 17, 18]. Two of these [17, 18] support only one-way matching of functional service descriptions to requestor queries as opposed to requirement-capability matching. They do not need to consider two-way matching since their focus is on matching functional aspects; when discussing purely functional requirements there is no functional requirement from the provider-side and no functional capabilities on the requestor-side. The third proposed matchmaking algorithm [2] performs requirement-capability matching for both sides. However, it does not take into account property attributes. Consequently, it will not support cases where both the requirement and capability point to the same concept but the concepts are annotated with different properties. For example, the requestor and provider may both use SSH (stated as a requirement on one side and a capability on the other), but if the requestor requires SSH using TripleDES and the provider is only capable of SSH with AES then these two should not match. Our matchmaker will perform requirement-capability matching, taking into account property annotations.

Specifically, when describing security information of resources, the ability to include properties in the matching algorithm is very important. This is due to the fact that security information, more so than functionality-related information can require detailed descriptions that make extensive use of properties. Complex statements can be made with multiple layers of properties. For example, there could be a security requirement that requires the use of XML-enc (*securityRequirement* property) with a symmetric encryption algorithm

(*hasAlgorithm* property) that has been declared a type 3 algorithm from the NSA (*hasNSALevel* property).

For the first task of the matchmaker, there are four possible levels of match for each requirement-capability pair: perfect match, close match, possible match, and no match in decreasing order of matching.

Perfect Match cases

Perfect matches occur when both one's capability and the other's requirement point to the same concept. The same concept can mean the exact same concept, or two concepts declared as equivalent in the ontology. There are two ways this can occur:

- Case 1. Both the requirement and capability specify the exact same ontology concept. The instances and property values specified by both sides are identical. This is the trivial case. For example, if a requestor query states that it requires the service to possess a VPN (Virtual Private Network) that possesses a Common Criteria EAL4 rating and a service describes its capability as possessing a VPN with a Common Criteria rating of EAL 4 then these two are a perfect match.
- Case 2. The requirement and capability refer to equivalent concepts, and if properties are specified, the properties are identical or equivalent. For example, a requestor's requirement specifies SSL and the provider's capability is listed as TLS. In the Main Security ontology, these two concepts are listed as equivalent classes; hence they are identical and will produce a perfect match. We sometimes call this an equivalence match to differentiate from the first case.

Close Match cases

A close match occurs when one's requirement is more general (i.e., described in less detail) than the other's capability. There are three ways this can occur:

- Case 1. The requirement specifies a more general concept at a higher level in the ontological hierarchy. For example, the requestor's capability is stated as DES while the provider's requirement asks for a symmetric encryption algorithm. DES is an instance of the 'SymmetricAlgorithm' class and thus lower in the hierarchy. We assume that the provider specified its requirement as a higher level concept because it does not care which specific algorithm is used as long as it is a symmetric encryption algorithm. Therefore, we can assume a match.
- Case 2. The requirement and capability have the same concept, but the capability is specified in more detail (i.e., property). For example, the requestor's capability is specified as AES with 256 bit keys while the provider's requirement asks for AES (with no properties). AES with 256 bit keys is a more specific instance of AES so we can assume that there is a match.
- Case 3. The requirement is stated in terms of a security objective while the capability is stated in terms of a security concept that supports that specific objective. For example, the requestor's requirement is stated as the objective of Confidentiality and the provider's capability is given as XML-Enc which has the *supportsSecurityObjective* value of Confidentiality. Since the requirement is looking for anything that supports Confidentiality and XML-Enc does support it, we view this as a match.

Possible Match cases

A possible match occurs when one's requirement is more specific (i.e., defined in more detail) than the other's capability. This is the opposite of a close match. A possible match does not rule out the possibility of a match, but the information available cannot ensure the capability can match the requirement. There are three ways this can occur:

- Case 1. The requirement specifies a more specific concept (lower in the hierarchy). For example, the requestor's capability is stated as symmetric encryption algorithm while the provider's requirement asks for DES. The symmetric encryption algorithm that the requestor is capable of could be DES, but it is not certain. Therefore, it is only a partial match.
- Case 2. The requirement and capability refer to the same concept, but the requirement specifies a more refined concept (i.e. property). For example, the capability is stated as AES while the requirement asks for AES with 256-bit keys. The AES specified in the capability could be possible of 256-bit key encryption, but it is not certain. Therefore, it is only a partial match.
- Case 3. The requirement is stated in terms of a security concept while the capability is stated in terms of a security objective that is supported by the security concept. For example, the requestor's requirement is stated as confidentiality while the provider's capability is stated as XML-Enc which supports confidentiality. The requestor may be capable of using XML-Enc, but it is not certain. All we can deduce is that the requestor is capable of confidentiality. Therefore, it is only a partial match.

No Match cases

No match occurs when one's capability and the other's requirement are disparate without the possibility of matching. There are two ways this can occur:

- Case 1. The requirement and capability point to two unrelated concepts. For example, the requirement states it requires DES and the capability states its capability as RSA. These concepts have no hierarchical relationship to each other and so are unrelated. There can be no match.
- Case 2. The requirement and capability point to the same concept but have different specifics (i.e. properties) with respect to that concept. For example, the requirement points to AES in CBC mode while the capability states AES in CFB mode. The capability and requirement can both use AES, but they require modes of operation; one is a block cipher the other is a stream cipher so they are not compatible.

For the second task of the matchmaker, it must attempt to match every requirement on one side against every capability on the other side. The degree of match for a single requirement is its highest level of match it has against all of the possible capabilities. The overall level of match between the requester and the provider is the same as the lowest degree of match of any of the requirement-capability pairs. There are four possibilities:

- If at least one of the requirements is not matched, then the requestor is not matched to the provider. The requestor will not be able to use the resource.
- If all the requirement-capability pairs are at least possible matches, then there is a possible match between the requester and the provider. This means there is not enough information to determine one way or the other whether the requester can use the

resource. Additional information or negotiation will be needed to make that determination.

- If all the requirement-capability pairs are at least close matches, then the requestor can indeed use the resource.
- If all the requirement-capability pairs are perfect matches, then obviously the requestor can use the resource.

In the following section, we will provide an example of the matching process between a service description and a query.

4.2. Application of the Matching Algorithm

In this section we examine how to actually describe services and create queries using the security ontologies, and how to find services using the matching algorithm. In our example, we have a service requestor looking for a book selling service. The service requestor would create queries to find services that match not only the desired functionality, but also the security capabilities and requirements of the requestor.

The following is an example of the requestor's security capabilities and requirements along with the part of their query that pertains to the security capability and requirements:

Requestor's Security Capabilities

1. Authentication via SAML with an X.509 Certificate signed by VeriSign

Requestor's Security Requirements

1. Authorization
2. SSH with the DES algorithm in CBC mode

```
<credential:X.509Certificate rdf:ID="X.509">
        <credential:issuer rdf:resource="VeriSign"/>
</credential:X.509Certificate>
<securityMain:SAML rdf:ID="Capability1">
        <securityMain:reqCredentials rdf:resource="&credential;X.509"/>
</securityMain:SAML>

<securityMain:Authorization rdf:ID="Requirement1"/>

<securityAlgorithms:DES rdf:ID= "Alg1">
        <securityAlgorithms:modesOfOperation rdf:resource="CBC"/>
</securityAlgorithms:DES>
<securityMain:SSH rdf:ID="Requirement2">
        <securityMain:hasEncryptionAlgorithm
        rdf:resource="&securityAlgorithms;Alg1"/>
</securityMain:SSH>

<agent:Agent rdf:about="#BookRequest">
        <securityCapability rdf:resource="#Capability1"/>
        <securityRequirement rdf:resource="#Requirement1"/>
        <securityRequirement rdf:resource="#Requirement2"/>
</agent>
```

On the other hand, a book selling service would create an OWL-S profile that includes its functional capabilities, as well as security requirements and capabilities. The following is the example security capability and requirement statements of the book selling service (BookSeller), along with the part of its OWL-Profile that would contain these statements.

BookSeller's Security Capabilities

1. SOAP Firewall with a Common Criteria level of EAL4
2. SSH with DES

BookSeller's Security Requirements

1. Authenticate via SAML with an X.509 Certificate

```
<securityMain:SOAPFirewall rdf:ID="Capability1">
        <securityMain:hasAssurance rdf:resource="&assurance;EAL4"/>
</securityMain:SOAPFirewall>

<securityMain:SSH rdf:ID="Capability2">
        <securityMain:hasEncryptionAlgorithm
        rdf:resource="&securityAlgorithms;DES"/>
</securityMain:SSH>

<credential:X.509Certificate rdf:ID="X.509"/>
<securityMain:SAML rdf:ID="Requirement1">
        <securityMain:reqCredentials rdf:resource="&credential;X.509"/>
</securityMain:SAML>

<profile:Profile rdf:about="#BookSeller1">
        <profile:serviceName>Book Seller 1</profile:serviceName>
        <profile:textDescription>This service sells all types of books
                </profile:textDescription>
        <securityCapability rdf:resource="#Capability1"/>
        <securityCapability rdf:resource="#Capability2"/>
        <securityRequirement rdf:resource="#Requirement1"/>
</profile:Profile>
```

Given this service description and the above query, the matching algorithm would match the requestor's capabilities to the provider's requirements and the requestor's requirements to the provider's capabilities in the following manner (Tables 2 and 3):

Table 2. Matching Requestor's Capabilities to Provider's Requirements

Requestor's Security Capability	**Provider's Security Requirement**	**Match Level**
Authentication via SAML with an X.509 Certificate signed by VeriSign	Authentication via SAML with an X.509 Certificate	Close Match

Table 3. Matching Requestor's Requirements to Provider's Capabilities

Requestor's Security Requirement	**Provider's SecurityCapability**	**Match Level**
Authorization	SOAP Firewall with Common Criteria level EAL4	Close Match
SSH with DES algorithm in CBC mode	SSH with DES algorithm	Possible Match

- In Table 2, the requestor's capability and the provider's requirement possess the same concepts, but the capability has more detail. This is Case 2 of the close match situation.
- In the first row of Table 3, the requestor's requirement was that a service provides Authorization. While the security objective of authorization is not explicitly stated in the OWL-S Profile of the provider, the reasoner was able to deduce that the SOAP Firewall supports authorization since it has a value of Authorization in its supportsSecurityObjective property. This is Case 3 of the close match situation.
- In the second row of Table 3, the requestor has a more detailed requirement regarding SSH than the provider has specified as its capability. This is Case 2 of the possible match situation. This could mean that either the provider cannot support the CBC mode of DES or it can support DES in CBC mode but decided not to provide this additional detail.

Since the lowest level of match in the three sets of requirement-capability pairs is possible match, the matchmaker will declare the service to be a possible match. The requester is not certain whether it can use the service. It must obtain additional information or negotiate with the provider to make that decision.

5. Conclusion and Future Work

Annotating resources with metadata enables them to be machine-understandable and facilitates automatic discovery and invocation. Most work in the area thus far has focused on annotation of resources in terms of functionality. However, security is an important issue especially in a network-centric environment. Most resources on the network are protected by some sort of security mechanisms. Satisfying functional requirements alone may not guarantee access to desired resources. As a result, annotation of resources in terms of security is just as important as annotation in terms of functionality.

In this paper, we presented the NRL Security Ontology for making security annotations. It is much more comprehensive than security ontologies previously available in terms of the number of concepts, the properties of the concepts, and the type of resources that can be described. Its organization is also more intuitive so that it is easier to use as well as to extend. New properties and instances can be added without modifying the overall class hierarchy. We demonstrated how the ontology can be applied to the context of Web services in a Service Oriented Architecture to describe security capabilities and requirements. A matchmaking algorithm was presented to perform requirement-capability matchmaking that takes into account not just the concepts, but also the properties of the concept. This is important because security annotations make extensive use of property attributes. The ability to take them into account makes this matching algorithm much more refined than previous work.

The creation of these ontologies is an iterative process. Additional instances and properties will always be needed to express new security statements. Classes and properties may be added and deleted as the security community continues to evaluate and refine the security ontologies. Additional ontologies are still needed to address issues such as privacy policies, access control, survivability, and QoS. We hope this work will serve as a catalyst in the development of standardized security-related ontologies with contributions from both the security community and the semantic Web community.

References

1. IA Architecture and Technical Framework (2004). Executive Summary of the End-to-End IA Component of the GIG Integrated Architecture, National Security Agency Information Assurance Directorate.
2. Denker, G., Kagal, L., Finin, T., Paolucci, M., and Sycara, K. (2003). Security for DAML Web Services: Annotation and Matchmaking. In *Proc. of the 2nd International Semantic Web Conference (ISWC2003)*: Sanibel Island, Florida.
3. Denker, G., Nguyen, S., and Ton, A. (2004). OWL-S Semantics of Security Web Services: a Case Study. In *1st European Semantic Web Symposium*: Heraklion, Greece.
4. Kagal, L., Paolucci, M., Srinivasan, N., Denker, G., Finin, T., and Sycara, K. (2004). Authorization and Privacy for Semantic Web Services. In *AAAI Spring Symposium, Workshop on Semantic Web Services*: Stanford, California.
5. W3C (2001). DAML+OIL (March 2001) Reference Description, http://www.w3.org/TR/daml+oil-reference.
6. W3C (2004). OWL Web Ontology Language Overview, http://www.w3.org/TR/owl-features/.
7. IETF and W3C Working Group (2001). XML Encryption, http://www.w3c.org/Encryption/2001.
8. IETF and W3C Working Group (2003). XML Signature, http://www.w3c.org/Signature.
9. OASIS SSTC (2005). Security Assertion Markup Language (SAML) 2.0 Technical Overview, Working Draft, http://www.oasis-open.org/committees/download.php/12938/sstc-saml-tech-overview-2.0-draft-06.pdf.
10. Noy, N.F., and McGuinness, D.L. (2001). Ontology Development 101: A Guide to Creating Your First Ontology, Stanford Knowledge Systems Laboratory, KSL-01-05.
11. W3C Recommendation (2004). OWL Web Ontology Language Guide, vol. 2005, W3C.
12. DAML Ontology Library. http://www.daml.org/ontologies/.
13. Schneier, B. (1996). Applied Cryptography, 2nd Edition (New York: John Wiley and Sons, Inc.).
14. Ferraiolo, D.F., Kuhn, D.R., and Chandramouli, R. (2003). Role-Based Access Control (Norwood, MA: Artech House).
15. Committee on National Security Systems (2003). National Information Assurance (IA) Glossary. pp. 85, http://www.cnss.gov/Assets/pdf/cnssi_4009.pdf: Ft. Meade, MD.
16. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., and Sycara, K. (2003). OWL-S: Semantic Markup for Web Services, http://www.daml.org/services/owl-s/1.1/overview/.
17. Jaeger, M., and Tang, S. (2004). Ranked Matching for Service Descriptions using DAML-S. In *Enterprise Modelling and Ontologies for Interoperability (EMOI), INTEROP 2004*: Riga, Latvia.

18. Srinivasan, N., Paolucci, M., and Sycara, K. (2004). Adding OWL-S to UDDI, Implementation and Throughput. In *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*: San Diego, California.
19. Gill, T. (2000). Metadata and the World Wide Web. In Introduction to Metadata: Pathways to Digital Information, 2nd Edition Edition, M. Baca, ed. (Los Angeles: Getty Information Institute).

APPENDIX A. Ontology vs. Taxonomy for Security Annotations

Metadata, literally 'data about data,' can include the content, context, and structure of the resource object. This metadata can be described using taxonomy or ontology. Although there is a tendency to use these two terms interchangeably, there are major differences between taxonomy and ontology. While a simple taxonomy can provide metadata to markup resources, taxonomies by themselves are limited in that they only describe and classify resources based on hierarchical (i.e., subclass) relationships among concepts. Ontology by definition is a "formal, shared conceptualization of a particular domain of interest" [19]. Ontology can express the entire knowledge domain, and provide a common vernacular to understand and express concepts. This knowledge is represented not only by subclass relationships, but also through properties that further define relationships, restrictions and interdependencies among concepts. We can provide better decision-ready information and contextual knowledge if we can mark these resources in such as way. In a sense, we are creating 'smart' data and 'smart' services with relevant properties that depict complex relationships and assert general facts about classes and specific facts about individuals.

In addition, ontologies enable class-based reasoning and deduction at the level of detail that taxonomies cannot provide. Ontologies provide us with the ability to reason about individuals and enable us to let individuals inherit properties by virtue of being a member of a certain class. For example, assume we had an ontology that described the type of credential required to authenticate oneself, and that we could use this ontology with a property called *reqCredential*. We can create an ontology rule stating that any resource that has CACcard (Common Access Card) as a value for the *reqCredential* property is a DoD resource. Then when searching for a resource, agents can deduce whether or not a particular resource is owned by the DoD by examining the value it contains in the *reqCredential* property.

We want to use ontologies to provide metadata to annotate the security requirements and capabilities of the requestor and the provider. Even now, with Web browsing, Web servers have their own security policies, security requirements and capabilities. For example, some Web sites require logging in or creating an account; others may require the use of SSL for secure communications; others may disclose personal information to third parties, etc. While current practices require humans to read the policies to figure out whether these meet our needs, in a dynamic, distributed environment, we expect our agents to act on our behalf to decipher and match the security requirements and capabilities of the resource to the requestor. Moreover, the requestor itself has its own set of security requirements and capabilities that have to be understood by the resource provider. This can all be realized through the use of security ontologies that let us express security requirements and capabilities, as well as access control policies, privacy policies, and QoS issues in a common, formal manner.

Some may argue that a simple taxonomy is sufficient to annotate resources with security-related information. However, there are cases where the simplicity of a hierarchical classification is not adequate to represent security-related information. For example, assume that we have a simple taxonomy such as the one in Figure A.1 to represent security

mechanisms. If we want to only express strict terms and concepts in our security statements, the taxonomy in Figure A.1 would be a sufficient method of doing so (e.g., this resource requires access control, that resource can provide an X.509 certificate, etc.). However, if we want to describe detailed concepts or compare security statements, there would not be a way to do so with the taxonomy. For example, if we wanted to state the following: "This resource requires authentication via an X.509v3 certificate that is signed specifically by VeriSign Certificate Authority, and it must still be valid." Then there is no way to express this using the strict hierarchical, classification-based notation of taxonomies. As another example, if we had a taxonomical representation of security algorithms, and wanted to state a capability such as "This resource has the capability to provide SSH version 3, and supports the AES encryption algorithm with a key length of 256 bits, and uses the Diffie-Hellman key exchange algorithm," it would also not be possible with the simple taxonomy. On the other hand, ontologies can provide rich, semantic meaning to describe a wide range of security statements. Additionally, we can specify properties and restrictions on properties that allow us to indicate restrictions to a particular class or individual of a class using ontologies. For example, one can state that a username-password pair must contain one and only one password using ontologies. Others may state that the service only accepts certificates that were created after a certain date. These things that are not possible to express with a taxonomy.

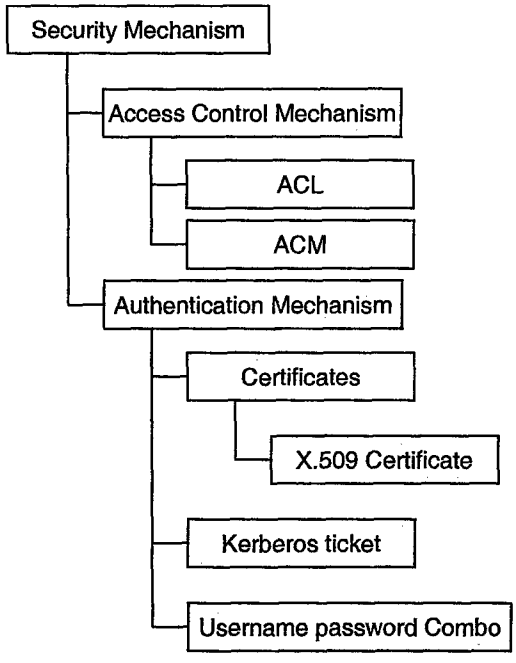


Figure A.1: Sample Taxonomical Representation of Selected Security Mechanisms

If we were to take the above sample security mechanism taxonomy and expand it into an ontology, it could be expressed as in Figure A.2 where some possible properties are added to a particular class, and are inherited by its subclasses and individuals, denoted by italicized phrases. For simplicity, property types and values are omitted, as they are somewhat self-explanatory. This is just a slight modification of the taxonomy, without using the full-fledged

expressive power of ontologies. Yet, even with the simple modifications of additions of properties, we can see that it now enables specification of more detailed security-related information. For example, we can now declare an instance of an X.509 Certificate class that would inherit all the properties from the X.509 Certificate class and its parent class, enabling expressions of version info, certificate authority, key value, and certificate owner if necessary.

Hence, it makes sense to use ontology for representing security concepts and annotating resources. In fact, due to the complex nature and relationships of various security concepts and the possible attributes they possess, ontologies are very well suited to express the security domain.

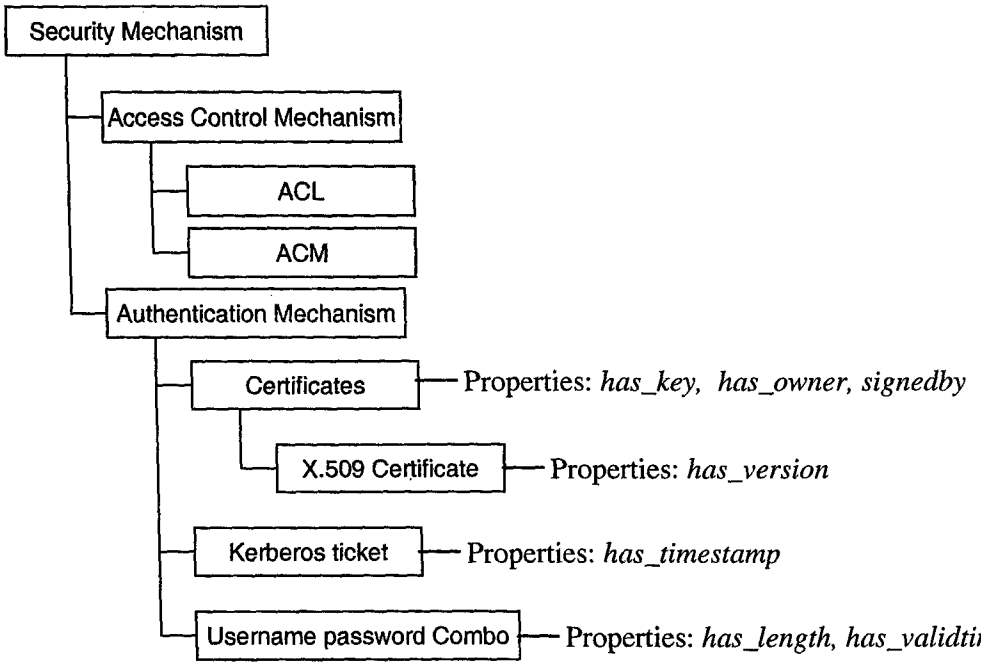


Figure A.2: Sample Ontological Representation with Some Properties Defined

APPENDIX B. Competency Questions for the Ontology

The competency questions devised to determine the scope of the ontology are stated as security requirements and capabilities for both the resource requestor and the provider.

B.1. Requestor-side security statements

A requestor may want have various security requirements it needs to be met and state its own capabilities that it possesses when searching for a resource.

Requirements

Requestors need the ability to specify the security features they desire from a provider. Possible types of security requirements could be:

1. Requirements specific to the exchange of data
    a. Resources must allow any inputs and outputs (all as a group or individually) to be encrypted, signed or protected in some manner with a specific algorithm or protocol
    b. Resources need to assure that sensitive (e.g., personal) data should be handled in an acceptable manner, according to a specific (privacy) policy or agreed upon way.
2. Requirements specific to security features
    a. Resources must posses specific security mechanism or can do a certain protocol, along with details such as particular algorithm or other properties
    b. Resources must meet certain security objectives (e.g., confidentiality, authentication)
3. Requirements specific to accessing the resource
    a. Resources can only belong to a specific entity (e.g., business or organization or individual or even country) and proof of ownership (i.e., what credentials are required) should be presented
    b. Resources can only come from entities that have a certain level of assurance or trust (i.e. rating method that shows level of trust other users or a trusted third party has assigned to it)

Capabilities

The requestor may also need to communicate what it security capabilities it possesses so that the provider allows access to the requestor, including:

1. Capabilities specific to the credentials it is capable of handling
    a. Resources can present credentials (of a given type) to the provider for authentication purposes
    b. Resources can obtain credentials if necessary from a third party
2. Capability regarding security features
    a. Resources can use security components such as security mechanisms, and security protocols that meet certain security objectives and have a given assurance level
    b. Resources can adhere to a specific security policy

B.2. Provider-side Security Statements

A resource provider may want to state the following security requirements and its capabilities of its resource. The types of statements made from the provider side would be mirror images of the requestor. Specifically, requestor-side capability statements would mirror provider-side requirements and visa versa.

Requirements

From the service perspective, security requirements are similar to the concept of policy. In this regards, providers want to express things like:

1. Requirements specific to authentication and access control
    a. Resources require only certain types of users (based on the credential they possess or association they belong to or even the location they are coming from) to access the service
    b. Resources require specific types of credentials from the requestor to access the resource
2. Requirements specific to the types of security features the requestor should possess
    a. Resources expect the requestor to be capable of certain security mechanisms or security protocols with various levels of detail

Capabilities

Provider capabilities are things that the resource provider can do for the requestor. Examples include:

1. Capabilities specific to the exchange of data
    a. Resources securely transmit input/output parameters (i.e. uses encryption or signing)
    b. Resources securely store data after it has been used
2. Capabilities specific to the security features that the provider possesses
    a. Resources can be used with given security mechanisms and protocols in various levels of detail
    b. Resources can support various security objectives
3. Capabilities specific to providing the resource
    a. Resources originate from given entity or location and this can be proven using given mechanism or credential
    b. Resources are rated or certified at given assurance level via specific organization with specified rating and this can be proven using certain credential

APPENDIX C. Graphical Representation of Security Ontologies

To aid in understanding the ontologies including the classes, properties and relationships, we provided a graphical representation of the set of NRL ontologies. In these diagrams and in the OWL files, we tried to maintain consistency in naming conventions for classes, properties and instances. The conventions we used are described briefly.

Classes are depicted as singular nouns or phrases and begin with an upper case letter. When a class is a combination of two words, such as SecurityProtocol, it is written as one word, but the beginning of each individual word is capitalized.

Properties begin with lowercase letters and are singular. They can be nouns or verbs. If the property name is more than one word, the rest of the words begin with capital letters, for example, *hasAssurance* and *isGovernmentStandard.*

Individuals also start with uppercase letters. In most cases, we follow the generally agreed upon notation of the instance to represent it. For example, IPSec is generally spelled with I, P and S in uppercase form, so we follow the notation when declaring IPSec as an instance. XML-enc is usually hyphenated, so we follow that form as well. For cases where an instance is two or more words, they are written as one word, such as CACCard.

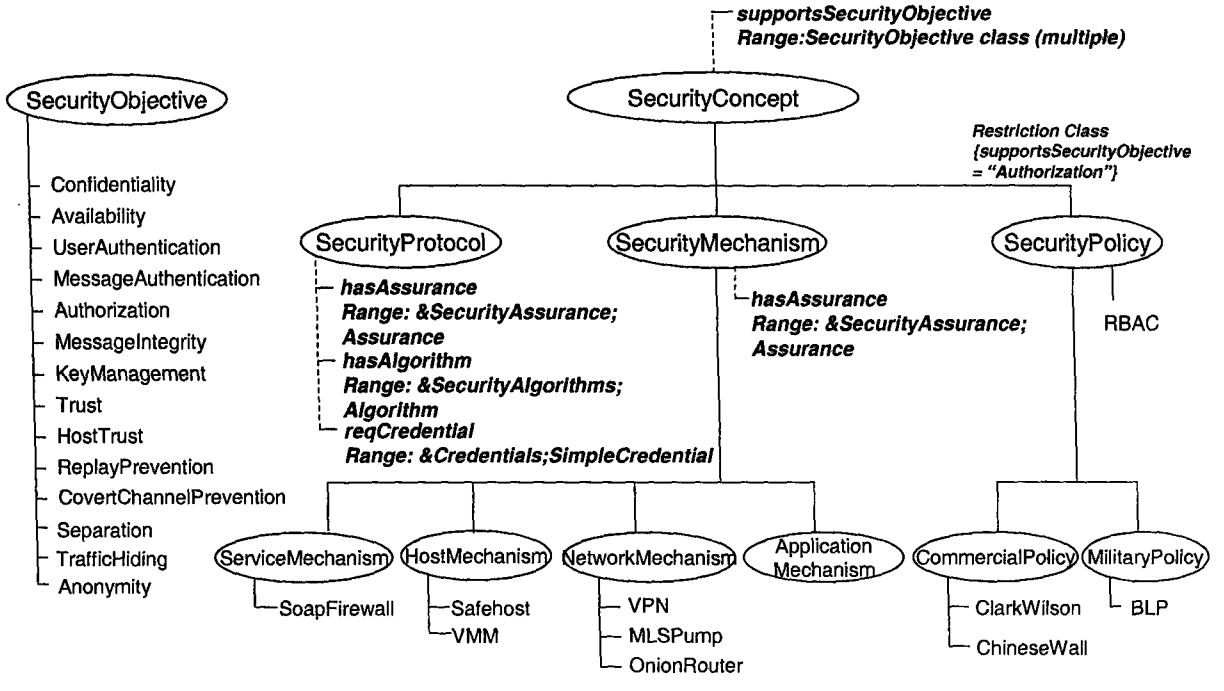
File names start with lowercase letters. In the case the file names are combinations of several words, the first letter of every word except the first word is capitalized. For example, securityAssurance.owl, and injObj.owl.

Trying to capture the complex relationships of the ontologies in a graphical form and still maintain readability required some brevity. Some of the notations we used to depict the ontology are:

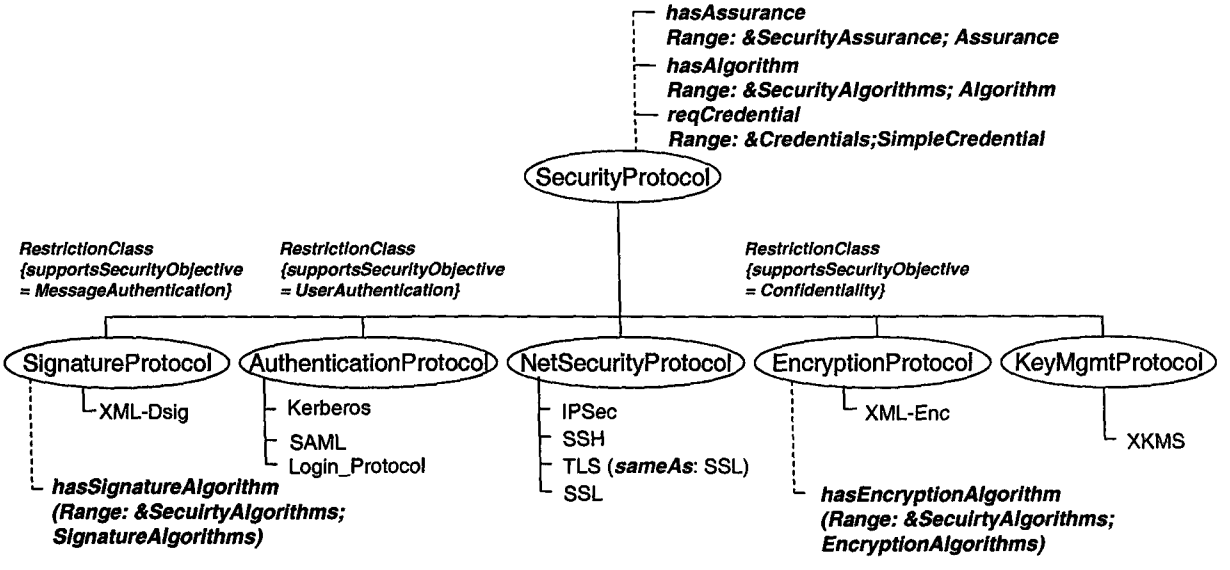
1. Classes are represented as (light blue) circles. Class names begin with uppercase letters.
2. Properties are represented as (blue) italicized and bold font. Property names always begin with lowercase letters. Properties whose ranges are not specified are data type properties meaning that their range can be literals (string, Boolean, integer, etc). For lack of space, the actual values of these data type properties were omitted, but can be seen in the OWL representations given in Appendix B.
3. Instances are represented in normal font, and begin with uppercase letters. They are connected to classes via a solid line, as opposed to properties which are connected to their respective classes via a dotted line.

C.1. NRL Security Main Ontology; securityMain.owl

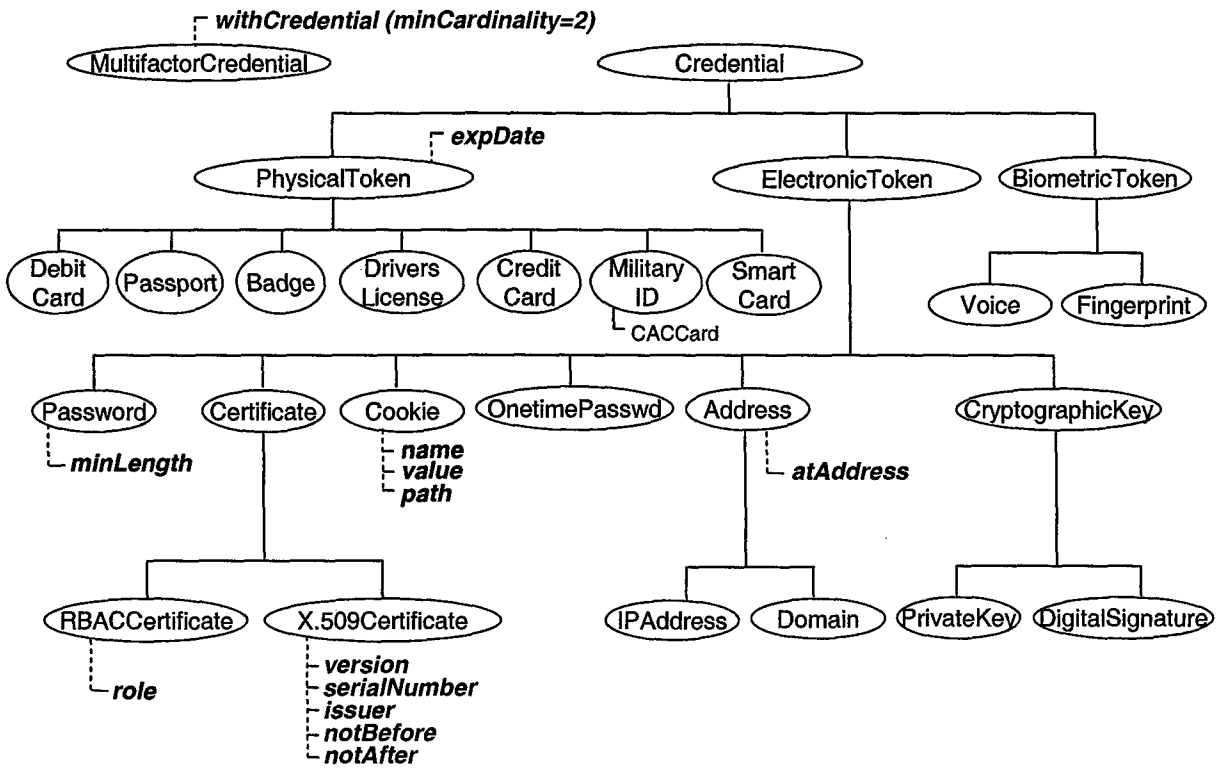
Our Main Security ontology was too big to fit in one picture, so the 'SecurityProtocol' subclass is actually depicted by itself, even though it is a part of the main ontology.



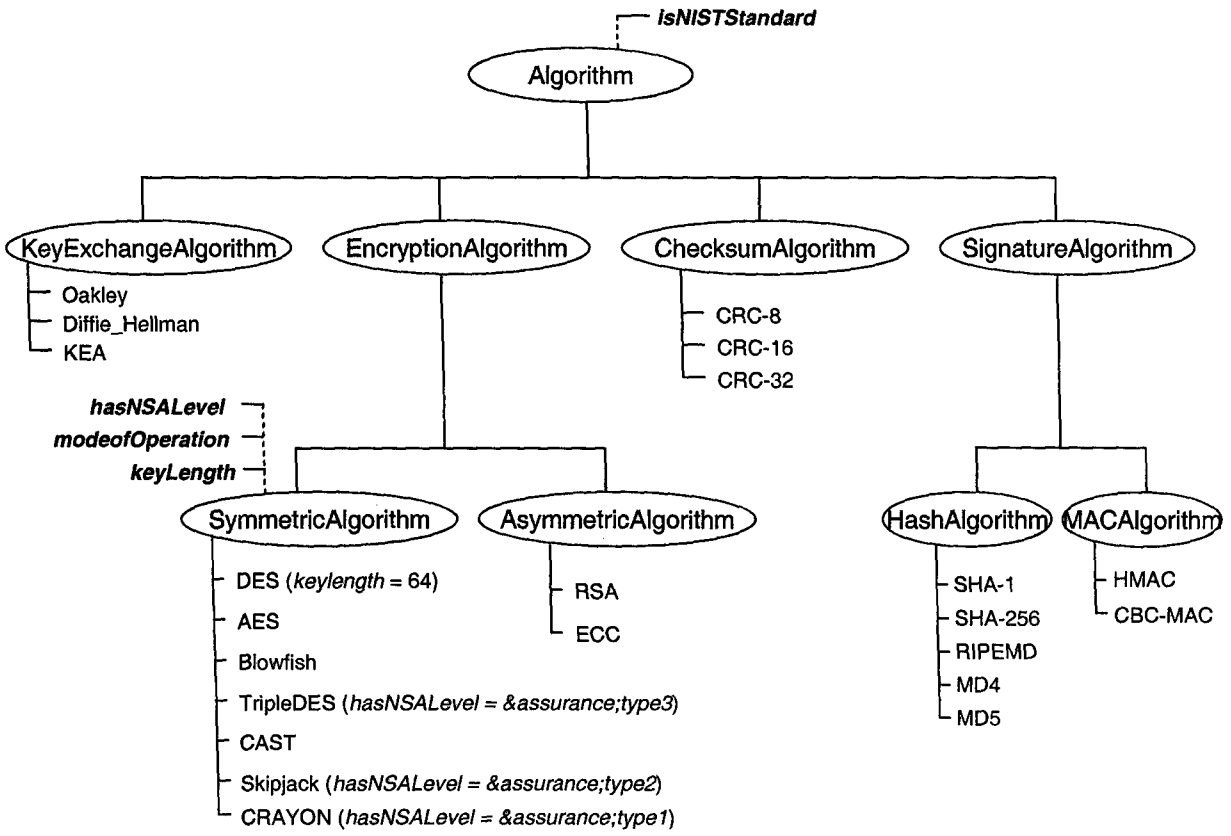
Security Main Ontology continued



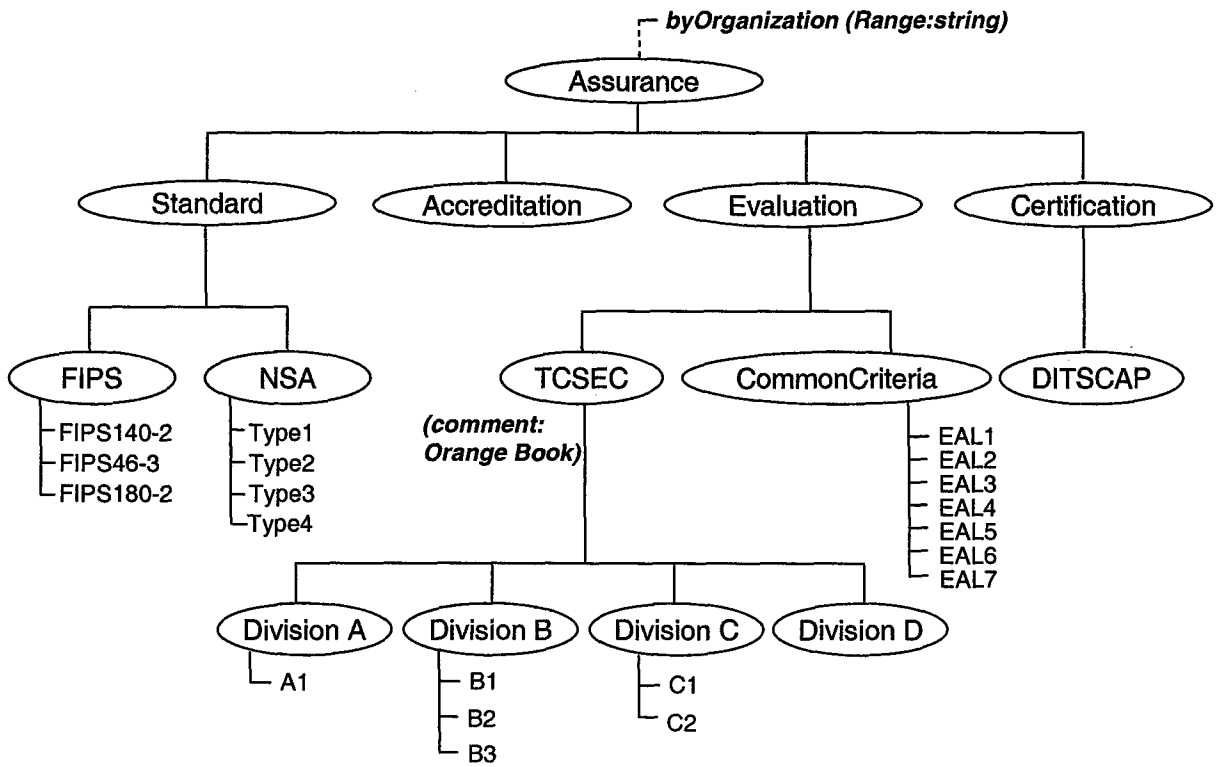
C.2. NRL Credentials Ontology; credentials.owl



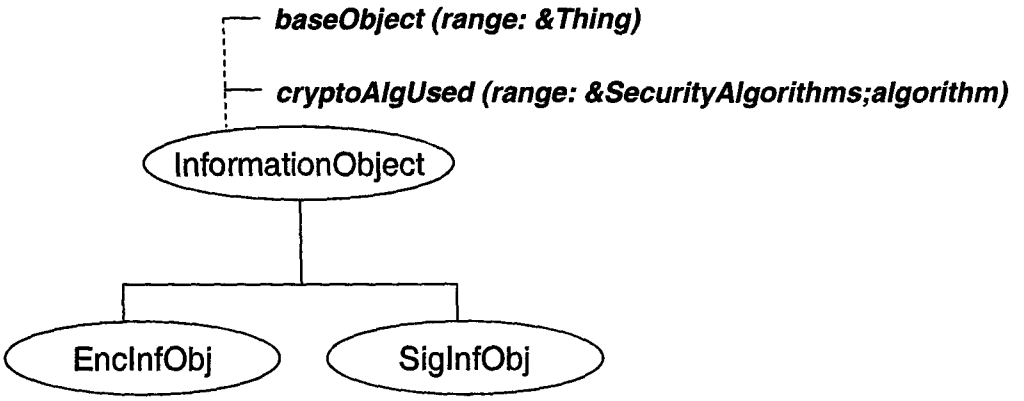
C.3. NRL Security Algorithms Ontology: securityAlgorithms.owl



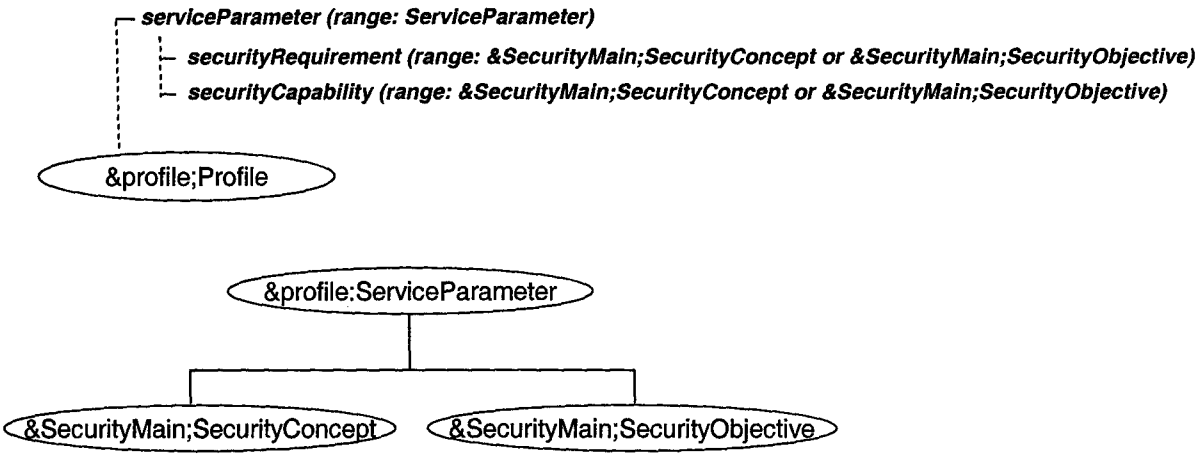
C.4. NRL Security Assurance Ontology: securityAssurance.owl



C.5. DAML infObj Ontology Modified; infObj.owl



C.6. DAML Service Security Ontology Modified; serviceSecurity.owl



C.7. DAML Agent Security Ontology Modified; agentSecurity.owl

APPENDIX D. OWL Representations of the NRL Security Ontology

D.1. Main Security Ontology (securityMain.owl)

```
<?xml version="1.0"?>

<!DOCTYPE uridef[
  <!ENTITY rdf            "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs           "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY owl            "http://www.w3.org/2002/07/owl#">
  <!ENTITY xsd            "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY credentials    "file://C:\NRLOntologyFiles\credentials.owl#">
  <!ENTITY algorithms     "file://C:\NRLOntologyFiles\securityAlgorithms.owl#">
  <!ENTITY assurance      "file://C:\NRLOntologyFiles\securityAssurance.owl#">
  <!ENTITY security       "file://C:\NRLOntologyFiles\securityMain.owl#">
  ]>

<rdf:RDF
    xmlns             ="&security;"
    xmlns:security    ="&security;"
    xml:base          ="&security;"
    xmlns:rdf         ="&rdf;"
    xmlns:rdfs        ="&rdfs;"
    xmlns:owl         ="&owl;"
    xmlns:xsd         ="&xsd;"
    xmlns:credentials ="&credentials;"
    xmlns:algorithms  ="&algorithms;"
    xmlns:assurance   ="&assurance;"
>

 <owl:Ontology>
   <rdfs:comment>
      A security ontology to annotate resources with security-related information
   </rdfs:comment>
 </owl:Ontology>

 <!-- Top Level Classes; SecurityObjective and SecurityConcept -->

  <owl:Class rdf:ID="SecurityObjective"/>

  <owl:Class rdf:ID="SecurityConcept"/>

 <!--
 Second Tier classes, subclasses of SecurityConcept;
 SecurityProtocol, SecurityMechansism, SecurityPolicy
 -->

  <owl:Class rdf:ID="SecurityProtocol">
    <rdfs:subClassOf rdf:resource="#SecurityConcept"/>
  </owl:Class>

  <owl:Class rdf:ID="SecurityMechanism">
    <rdfs:subClassOf rdf:resource="#SecurityConcept"/>
  </owl:Class>

  <owl:Class rdf:ID="SecurityPolicy">
    <rdfs:subClassOf rdf:resource="#SecurityConcept"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="supportsSecurityObjectives"/>
        <owl:hasValue rdf:resource="Authorization"/>
      </owl:Restriction>
    </rdfs:subClassOf>
```

```
 </owl:Class>

<!--
      Subclasses of SecurityProtocol class, including
      SignatureProtocol, AuthenticationProtocol, NetworkSecurityProtocol,
      EncryptionProtocol, and KeyMgmtProtocol
-->

 <owl:Class rdf:ID="AuthenticationProtocol">
   <rdfs:subClassOf rdf:resource="#SecurityProtocol"/>
   <rdfs:subClassOf>
     <owl:Restriction>
       <owl:onProperty rdf:resource="#supportsSecurityObjectives"/>
       <owl:hasValue rdf:resource="#UserAuthentication"/>
     </owl:Restriction>
   </rdfs:subClassOf>
 </owl:Class>

 <owl:Class rdf:ID="KeyManagementProtocol">
   <rdfs:subClassOf rdf:resource="#SecurityProtocol"/>
   <rdfs:subClassOf>
     <owl:Restriction>
       <owl:onProperty rdf:resource="#supportsSecurityObjectives"/>
       <owl:hasValue rdf:resource="#KeyManagement"/>
     </owl:Restriction>
   </rdfs:subClassOf>
 </owl:Class>

 <owl:Class rdf:ID="EncryptionProtocol">
   <rdfs:subClassOf rdf:resource="#SecurityProtocol"/>
   <rdfs:subClassOf>
     <owl:Restriction>
       <owl:onProperty rdf:resource="#supportsSecurityObjectives"/>
       <owl:hasValue rdf:resource="#Confidentiality"/>
     </owl:Restriction>
   </rdfs:subClassOf>
 </owl:Class>

 <owl:Class rdf:ID="SignatureProtocol">
   <rdfs:subClassOf rdf:resource="#SecurityProtocol"/>
   <rdfs:subClassOf>
     <owl:Restriction>
       <owl:onProperty rdf:resource="#supportsSecurityObjectives"/>
       <owl:hasValue rdf:resource="#MessageAuthentication"/>
     </owl:Restriction>
   </rdfs:subClassOf>
 </owl:Class>

 <owl:Class rdf:ID="NetworkSecurityProtocol">
   <rdfs:subClassOf rdf:resource="#SecurityProtocol"/>
 </owl:Class>

 <!--
       Subclasses of SecurityMechanism class; Service, Host, Network, Application
 -->

 <owl:Class rdf:ID="ServiceMechanism">
   <rdfs:subClassOf rdf:resource="#SecurityMechanism"/>
 </owl:Class>

 <owl:Class rdf:ID="HostMechanism">
   <rdfs:subClassOf rdf:resource="#SecurityMechanism"/>
 </owl:Class>

 <owl:Class rdf:ID="NetworkMechanism">
   <rdfs:subClassOf rdf:resource="#SecurityMechanism"/>
 </owl:Class>

 <owl:Class rdf:ID="ApplicationMechanism">
   <rdfs:subClassOf rdf:resource="#SecurityMechanism"/>
 </owl:Class>
```

```
<!--
     Subclasses of SecurityPolicy class; Commercial, Military
-->

<owl:Class rdf:ID="CommercialPolicy">
  <rdfs:subClassOf rdf:resource="#SecurityPolicy"/>
</owl:Class>

<owl:Class rdf:ID="MilitaryPolicy">
  <rdfs:subClassOf rdf:resource="#SecurityPolicy"/>
</owl:Class>

<!--
     Property Definitions: SecurityConcept class
-->

<owl:ObjectProperty rdf:ID="supportsSecurityObjectives">
  <rdfs:domain rdf:resource="#SecurityConcept"/>
  <rdfs:range rdf:resource="#SecurityObjective"/>
  <rdfs:comment>
    Any SecurityConcept can support one or more of the Security Objectives defined
in the SecurityObjective class
  </rdfs:comment>
</owl:ObjectProperty>

<!--
Property Definitions: SecurityProtocol class
-->
<owl:ObjectProperty rdf:ID="hasAlgorithm">
  <rdfs:domain rdf:resource="#SecurityProtocol"/>
  <rdfs:range rdf:resource="&algorithms;Algorithm"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasAssurance">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#SecurityProtocol"/>
        <owl:Class rdf:about="#SecurityMechanism"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="&assurance;Assurance"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="reqCredential">
  <rdfs:domain rdf:resource="#SecurityProtocol"/>
  <rdfs:range rdf:resource="&credentials;Credential"/>
</owl:ObjectProperty>

<!--
Property Definitions: SecurityMechanism class
-->
<!-- Property is hasAssurance, which is already declared above -->

<!--
Property Definitions: SignatureProtocol class
-->
<owl:ObjectProperty rdf:ID="hasSignatureAlgorithm">
  <rdfs:subPropertyOf rdf:resource="#hasAlgorithm"/>
  <rdfs:range rdf:resource="&algorithms;SignatureAlgorithm"/>
</owl:ObjectProperty>

<!--
Property Definitions: EncryptionProtocol class
-->
<owl:ObjectProperty rdf:ID="hasEncryptionAlgorithm">
  <rdfs:subPropertyOf rdf:resource="#hasAlgorithm"/>
  <rdfs:range rdf:resource="&algorithms;EncryptionAlgorithm"/>
</owl:ObjectProperty>

<!--
```

```
  Instance declarations for SecurityObjective class
  -->

  <SecurityObjective rdf:ID="Confidentiality">
    <rdfs:comment>
      Protects against information being disclosed or revealed to unauthorized
parties.
    </rdfs:comment>
  </SecurityObjective>

  <SecurityObjective rdf:ID="Availability">
    <rdfs:comment>
      Provides guarantee that resource is available to authorized users at any
authorized time.
    </rdfs:comment>
  </SecurityObjective>

  <SecurityObjective rdf:ID="UserAuthentication">
    <rdfs:comment>
      Provides assurance of the identity of a person or entity.
    </rdfs:comment>
  </SecurityObjective>

  <SecurityObjective rdf:ID="MessageAuthentication">
    <rdfs:comment>
      Provides guarantee that message came from whoever claimed to have sent it.
    </rdfs:comment>
  </SecurityObjective>

  <SecurityObjective rdf:ID="Authorization">
    <rdfs:comment>
      Only authorized entities are allowed to access resources in an authorized
manner.
      (Aka Access Control). If desired, user may create instance of AccessControl as
well.
    </rdfs:comment>
  </SecurityObjective>

  <SecurityObjective rdf:ID="MessageIntegrity">
    <rdfs:comment>
      Data integrity or Message integrity; data cannot be changed, deleted,
modified, etc by unauthorized parties.
    </rdfs:comment>
  </SecurityObjective>

  <SecurityObjective rdf:ID="KeyManagement">
    <rdfs:comment>
      (SecondaryObjective) Securely manages keys for legitimate users.
    </rdfs:comment>
  </SecurityObjective>

  <SecurityObjective rdf:ID="ReplayPrevention">
    <rdfs:comment>
      (Secondary Objective) Protect against replay attacks
    </rdfs:comment>
  </SecurityObjective>

  <SecurityObjective rdf:ID="Trust">
    <rdfs:comment>
      (Secondary Objective) How to trust entity. Trust not only of identification,
      but statements, claims, etc.
    </rdfs:comment>
  </SecurityObjective>

  <SecurityObjective rdf:ID="HostTrust">
    <rdfs:comment>
      (Secondary Objective) How to trust host or platform. Is relevant to Common
criteria, etc.
    </rdfs:comment>
  </SecurityObjective>

  <SecurityObjective rdf:ID="CovertChannelPrevention">
```

```
  <rdfs:comment>
    (Secondary Objective) Prevent or limit the existance of covert channels
  </rdfs:comment>
</SecurityObjective>

<SecurityObjective rdf:ID="Separation">
  <rdfs:comment>
    (Secondary Objective)
  </rdfs:comment>
</SecurityObjective>

<SecurityObjective rdf:ID="TrafficHiding">
  <rdfs:comment>
    (Secondary Objective) Pad traffic with bogus data to hide traffic patterns
  </rdfs:comment>
</SecurityObjective>

<SecurityObjective rdf:ID="Anonymity">
  <rdfs:comment>
    (Secondary Objective) Provides anonymity
  </rdfs:comment>
</SecurityObjective>

<!--
Instance declarations for AuthenticationProtocol class
-->
<AuthenticationProtocol rdf:ID="Kerberos"/>

<AuthenticationProtocol rdf:ID="SAML"/>

<AuthenticationProtocol rdf:ID="Login_Protocol"/>

<!--
Instance declarations for SignatureProtocol class
-->
<SignatureProtocol rdf:ID="XML-dsig"/>

<!--
Instance declarations for EncryptionProtocol class
-->
<EncryptionProtocol rdf:ID="XML-enc"/>

<!--
Instance declarations for KeyManagementProtocol class
-->

<KeyManagementProtocol rdf:ID="XKMS"/>

<!--
Instance declarations for NetworkSecurityProtocol class
-->
<NetworkSecurityProtocol rdf:ID="SSH">
  <supportsSecurityObjectives rdf:resource="#Integrity"/>
  <supportsSecurityObjectives rdf:resource="#Confidentiality"/>
  <supportsSecurityObjectives rdf:resource="#UserAuthentication"/>
  <supportsSecurityObjectives rdf:resource="#MessageAuthentication"/>
</NetworkSecurityProtocol>

<NetworkSecurityProtocol rdf:ID="IPSec">
  <supportsSecurityObjectives rdf:resource="#MessageAuthentication"/>
  <supportsSecurityObjectives rdf:resource="#Integrity"/>
  <supportsSecurityObjectives rdf:resource="#Confidentiality"/>
</NetworkSecurityProtocol>

<NetworkSecurityProtocol rdf:ID="SSL">
  <owl:sameAs rdf:resource="#TLS"/>
  <supportsSecurityObjectives rdf:resource="#Confidentiality"/>
  <supportsSecurityObjectives rdf:resource="#UserAuthentication"/>
</NetworkSecurityProtocol>

<NetworkSecurityProtocol rdf:ID="TLS">
  <owl:sameAs rdf:resource="#SSL"/>
```

```
  <supportsSecurityObjectives rdf:resource="#UserAuthentication"/>
  <supportsSecurityObjectives rdf:resource="#Confidentiality"/>
</NetworkSecurityProtocol>

<!--
Instance declarations for SecurityMechanism:ServiceMechanism class
-->

<ServiceMechanism  rdf:ID="SoapFirewall">
  <supportsSecurityObjectives rdf:resource="#Authorization"/>
</ServiceMechanism>

<!--
Instance declarations for SecurityMechanism:HostMechanism class
-->

<HostMechanism rdf:ID="Safehost">
  <supportsSecurityObjectives rdf:resource="#HostTrust"/>
</HostMechanism>

<HostMechanism rdf:ID="VMM">
  <supportsSecurityObjectives rdf:resource="#HostTrust"/>
</HostMechanism>

<!--
Instance declarations for SecurityMechanism:NetworkMechanism class
-->

<NetworkMechanism rdf:ID="VPN">
  <supportsSecurityObjectives rdf:resource="#Confidentiality"/>
  <supportsSecurityObjectives rdf:resource="#Separation"/>
</NetworkMechanism>

<NetworkMechanism rdf:ID="MLS_Pump">
  <supportsSecurityObjectives rdf:resource="#CovertChannelPrevention"/>
</NetworkMechanism>

<NetworkMechanism rdf:ID="OnionRouter">
  <supportsSecurityObjectives rdf:resource="#Anonymity"/>
  <supportsSecurityObjectives rdf:resource="#Confidentiality"/>
</NetworkMechanism>

<!--
Instance declarations for SecurityPolicy class
-->

<SecurityPolicy rdf:ID="RBAC">
  <rdfs:comment>
    Role-based Access Control
  </rdfs:comment>
</SecurityPolicy>

<!--
Instance declarations for SecurityPolicy:Commercial class
-->
<CommercialPolicy rdf:ID="ClarkWilson"/>

<CommercialPolicy rdf:ID="ChineseWall"/>

<!--
Instance declarations for SecurityPolicy:Military class
-->
<MilitaryPolicy rdf:ID="BLP">
  <rdfs:comment>
    Bell La-Padula
  </rdfs:comment>
</MilitaryPolicy>

</rdf:RDF>
```

D.2. Credentials Ontology (credentials.owl)

```
<?xml version="1.0"?>
<!DOCTYPE uridef[
  <!ENTITY rdf                    "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs           "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY owl                    "http://www.w3.org/2002/07/owl#">
  <!ENTITY xsd                    "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY credentials       "file://C:\NRLOntologyFiles\credentials.owl#">
  <!ENTITY algorithms        "file://C:\NRLOntologyFiles\securityAlgorithms.owl#">
  <!ENTITY assurance         "file://C:\NRLOntologyFiles\securityAssurance.owl#">
  <!ENTITY security   "file://C:\NRLOntologyFiles\securityMain.owl#">
  ]>

<rdf:RDF
   xmlns              ="&credentials;"
   xmlns:credentials  ="&credentials;"
   xml:base           ="&credentials;"
   xmlns:rdf          ="&rdf;"
   xmlns:rdfs         ="&rdfs;"
   xmlns:owl          ="&owl;"
   xmlns:xsd          ="&xsd;"
   xmlns:security     ="security;"
   xmlns:algorithms   ="&algorithms;"
   xmlns:assurance    ="&assurance;"
>

 <owl:Ontology>
   <rdfs:comment>
      A credential ontology to describe authentication credentials.
   </rdfs:comment>
 </owl:Ontology>

  <!-- Top Level Class; Credential Class
  -->

  <owl:Class rdf:ID="Credential"/>

  <!--
  Subclasses of Credential: PhysicalToken class, ElectronicToken class,
  BiometricToken class
  -->

  <owl:Class rdf:ID="PhysicalToken">
    <rdfs:subClassOf rdf:resource="#Credential"/>
  </owl:Class>

  <owl:Class rdf:ID="ElectronicToken">
    <rdfs:subClassOf rdf:resource="#Credential"/>
  </owl:Class>

  <owl:Class rdf:ID="BiometricToken">
    <rdfs:subClassOf rdf:resource="#Credential"/>
  </owl:Class>

  <!--
  Subclass definitions of the PhysicalToken class
  Most of these subclasses are taken from the SRI Credential ontology
  -->

  <owl:Class rdf:ID="Badge">
    <rdfs:subClassOf rdf:resource="#PhysicalToken"/>
    <rdfs:comment>such as a police badge</rdfs:comment>
  </owl:Class>

  <owl:Class rdf:ID="DebitCard">
    <rdfs:subClassOf rdf:resource="#PhysicalToken"/>
  </owl:Class>
```

```
<owl:Class rdf:ID="CreditCard">
  <rdfs:subClassOf rdf:resource="#PhysicalToken"/>
</owl:Class>

<owl:Class rdf:ID="SmartCard">
  <rdfs:subClassOf rdf:resource="#PhysicalToken"/>
</owl:Class>

<owl:Class rdf:ID="Passport">
  <rdfs:subClassOf rdf:resource="#PhysicalToken"/>
</owl:Class>

<owl:Class rdf:ID="DriversLicense">
  <rdfs:subClassOf rdf:resource="#PhysicalToken"/>
</owl:Class>

<owl:Class rdf:ID="MilitaryID">
  <rdfs:subClassOf rdf:resource="#PhysicalToken"/>
</owl:Class>

<!--
Subclass defintions of the ElectronicToken class
-->

<owl:Class rdf:ID="Address">
  <rdfs:subClassOf rdf:resource="#ElectronicToken"/>
</owl:Class>

<owl:Class rdf:ID="Password">
  <rdfs:subClassOf rdf:resource="#ElectronicToken"/>
</owl:Class>

<owl:Class rdf:ID="OneTimePassword">
  <rdfs:subClassOf rdf:resource="#ElectronicToken"/>
</owl:Class>

<owl:Class rdf:id="Certificate">
  <rdfs:subClassOf rdf:resource="#ElectronicToken"/>
</owl:Class>

<owl:Class rdf:ID="Cookie">
  <rdfs:subClassOf rdf:resource="#ElectronicToken"/>
</owl:Class>

<owl:Class rdf:ID="CryptographicKey">
  <rdfs:subClassOf rdf:resource="#ElectronicToken"/>
</owl:Class>

<!--
Subclass defintions of the BiometricToken class; Voice and Fingerprint classes
-->

<owl:Class rdf:ID="Voice">
  <rdfs:subClassOf rdf:resource="#BiometricToken"/>
</owl:Class>

<owl:Class rdf:ID="Fingerprint">
  <rdfs:subClassOf rdf:resource="#BiometricToken"/>
</owl:Class>

<!--
Subclass definitions of the Address Class; IPAddress and Domain
-->

<owl:Class rdf:ID="IPAddress">
  <rdfs:subClassOf rdf:resource="#Address"/>
</owl:Class>

<owl:Class rdf:ID="Domain">
  <rdfs:subClassOf rdf:resource="#Address"/>
</owl:Class>
```

```
<!--
Subclass definitions of the Certificate class
-->

<owl:Class rdf:ID="X.509Certificate">
  <rdfs:subClassOf rdf:resource="#Certificate"/>
</owl:Class>

<owl:Class rdf:ID="RBACCertificate">
  <rdfs:subClassOf rdf:resource="#Certificate"/>
</owl:Class>

<!--
Subclass Definitions of CryptographicKey class; DigitalSignature and PrivateKey
classes
-->

<owl:Class rdf:ID="DigitalSignature">
  <rdfs:subClassOf rdf:resource="#CryptographicKey"/>
</owl:Class>

<owl:Class rdf:ID="PrivateKey">
  <rdfs:subClassOf rdf:resource="#CryptographicKey"/>
</owl:Class>

<!--
  MultifactorCredential is a Separate class to describe n-factor authentication;
  where n different credentials are required
  (e.g. two factor authentication with smart card and PIN)
  Types of credentials are defined through a property, "withCredential".
  Because it is multifactor, it requires at least two types of credentials
  enforced thru the minCardinality property
-->
<owl:Class rdf:ID="MultifactorCredential">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#withCredential1"/>
      <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">2</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<!--
  Property Definitions
-->

<!-- Property Definition for PhysicalToken class -->
<owl:DatatypeProperty rdf:ID="expDate">
  <rdfs:domain rdf:resource="#PhysicalToken"/>
  <rdfs:range rdf:resource="&xsd;dateTime"/>
</owl:DatatypeProperty>

<!-- Property Definition for Password class -->
<owl:DatatypeProperty rdf:ID="minLength">
  <rdfs:domain rdf:resource="#Password"/>
  <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>

<!-- Property Definition for Cookie class.
      These follow SRI ontology's properties for its cookie class
-->

<owl:DatatypeProperty rdf:ID="path">
  <rdfs:domain rdf:resource="#Cookie"/>
  <rdfs:range rdf:resource="&xsd;anyURI"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="value">
  <rdfs:range rdf:resource="&xsd;string"/>
  <rdfs:domain rdf:resource="#Cookie"/>
</owl:DatatypeProperty>
```

```
<owl:DatatypeProperty rdf:ID="name">
  <rdfs:range rdf:resource="&xsd;string"/>
  <rdfs:domain rdf:resource="#Cookie"/>
</owl:DatatypeProperty>

<!-- Property Definition for Address class -->

<owl:DatatypeProperty rdf:ID="atAddress">
  <rdfs:domain rdf:resource="#Address"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<!-- Property Definition for RBAC Certificate class -->
<owl:DatatypeProperty rdf:ID="role">
  <rdfs:domain rdf:resource="#RBACCertificate"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<!-- Property Definitions for X.509Certificate class
      This includes version, serialNumber, issuer, notBefore and notAfter
-->

<owl:DatatypeProperty rdf:ID="version">
  <rdfs:domain rdf:resource="#X.509Certificate"/>
  <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="serialNumber">
  <rdfs:domain rdf:resource="#X.509Certificate"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="issuer">
  <rdfs:domain rdf:resource="#X.509Certificate"/>
  <rdfs:range rdf:resource="&xsd;string"/>
  <rdfs:comment>this is the certificate authority</rdfs:comment>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="notBefore">
  <rdfs:domain rdf:resource="#X.509Certificate"/>
  <rdfs:range rdf:resource="&xsd;dateTime"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="notAfter">
  <rdfs:domain rdf:resource="#X.509Certificate"/>
  <rdfs:range rdf:resource="&xsd;dateTime"/>
</owl:DatatypeProperty>

<!-- Property definition for the MultifactorCredential class -->
<owl:ObjectProperty rdf:ID="withCredential">
  <rdfs:domain rdf:resource="#MultifactorCredential"/>
  <rdfs:range rdf:resource="#Credential"/>
</owl:ObjectProperty>

<!--
Instances
-->
<MilitaryID rdf:ID="CACCard"/>

</rdf:RDF>
```

D.3. Security Algorithms Ontology (securityAlgorithms.owl)

```
<?xml version="1.0"?>

<!DOCTYPE uridef[
  <!ENTITY rdf                    "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs            "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY owl                    "http://www.w3.org/2002/07/owl#">
  <!ENTITY xsd                    "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY security          "file://C:\NRLOntologyFiles\securityMain.owl#">
  <!ENTITY algorithms        "file://C:\NRLOntologyFiles\securityAlgorithms.owl#">
  <!ENTITY assurance         "file://C:\NRLOntologyFiles\securityAssurance.owl#">
  ]>

<rdf:RDF
    xmlns              ="&algorithms;"
    xmlns:algorithms   ="&algorithms;"
    xml:base           ="&algorithms;"
    xmlns:rdf          ="&rdf;"
    xmlns:rdfs         ="&rdfs;"
    xmlns:owl          ="&owl;"
    xmlns:xsd          ="&xsd;"
    xmlns:security     ="&security;"
    xmlns:assurance    ="&assurance;"
>

  <owl:Ontology>
     <rdfs:comment>
       An ontology to describe various cryptographic algorithms
     </rdfs:comment>
  </owl:Ontology>

  <!--
  Top Class Definition: Algorithm
  -->

  <owl:Class rdf:ID="Algorithm">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#isNISTStandard"/>
        <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <!--
  Subclass definitions of Algorithm Class; KeyExchangeAlgorithm, Encryption
  Algorithms, SignatureAlgorithm, and ChecksumAlgorithm
  -->

  <owl:Class rdf:ID="KeyExchangeAlgorithm">
    <rdfs:subClassOf rdf:resource="#Algorithm"/>
  </owl:Class>

  <owl:Class rdf:ID="EncryptionAlgorithm">
    <rdfs:subClassOf rdf:resource="#Algorithm"/>
  </owl:Class>

  <owl:Class rdf:ID="SignatureAlgorithm">
    <rdfs:subClassOf rdf:resource="#Algorithm"/>
```

```
</owl:Class>

<owl:Class rdf:ID="ChecksumAlgorithm">
  <rdfs:subClassOf rdf:resource="#Algorithm"/>
</owl:Class>

<!--
Subclasses of Encryption Algorithm: SymmetricAlgorithm and AsymmetricAlgorithm
-->

<owl:Class rdf:ID="SymmetricAlgorithm">
  <rdfs:subClassOf rdf:resource="#EncryptionAlgorithm"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="hasNSALevel"/>
      <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="AsymmetricAlgorithm">
  <rdfs:subClassOf rdf:resource="#EncryptionAlgorithm"/>
</owl:Class>

<!--
Subclasses of SignatureAlgorithm class: HashAlgorithm and MACAlgorithm
-->

<owl:Class rdf:ID="HashAlgorithm">
  <rdfs:subClassOf rdf:resource="#SignatureAlgorithm"/>
  <rdfs:comment>
    Key dependent function; useful in providing user authentication without a
    secret key
  </rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="MACAlgorithm">
  <rdfs:subClassOf rdf:resource="#SignatureAlgorithm"/>
</owl:Class>

<!-- Property Definitions -->

<!-- Property Definition for the Algorithm Class -->

<owl:DatatypeProperty rdf:ID="isNISTStandard">
  <rdfs:domain rdf:resource="#Algorithm"/>
  <rdfs:range rdf:resource="&xsd;boolean"/>
  <rdfs:comment>NIST FIPS Standard</rdfs:comment>
</owl:DatatypeProperty>

<!-- Property Defintions for the SymmetricAlgorithm class -->

<owl:DatatypeProperty rdf:ID="keyLength">
  <rdfs:domain rdf:resource="#SymmetricAlgorithm"/>
  <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="modeOfOperation">
  <rdfs:domain rdf:resource="#SymmetricAlgorithm"/>
  <rdfs:range rdf:resource="&xsd;string"/>
  <rdfs:comment>Can be CBC, ECB, OFB, CFB, or Counter</rdfs:comment>
</owl:DatatypeProperty>
```

```
  <owl:ObjectProperty rdf:id="hasNSALevel">
    <rdfs:domain rdf:resource="#SymmetricAlgorithm"/>
    <rdfs:range rdf:resource="&assurance;NSA"/>
    <rdfs:comment>These can be Type1, Type 2, Type3, or Type4</rdfs:comment>
  </owl:ObjectProperty>

  <!-- Instances -->

  <KeyExchangeAlgorithm rdf:ID="Diffie_Hellman"/>
  <KeyExchangeAlgorithm rdf:ID="KEA"/>
  <KeyExchangeAlgorithm rdf:ID="Oakley"/>

  <ChecksumAlgorithm rdf:ID="CRC-8"/>
  <ChecksumAlgorithm rdf:ID="CRC-16"/>
  <ChecksumAlgorithm rdf:ID="CRC-32"/>

  <SymmetricAlgorithm rdf:ID="AES">
    <hasNSALevel rdf:resource="&assurance;Type3"/>
  </SymmetricAlgorithm>

  <!-- Instances for the SymmetricAlgorithm class -->

  <SymmetricAlgorithm rdf:ID="RC4"/>

  <SymmetricAlgorithm rdf:ID="Skipjack">
    <hasNSALevel rdf:resource="&assurance;Type2"/>
  </SymmetricAlgorithm>

  <SymmetricAlgorithm rdf:ID="CRAYON">
    <hasNSALevel rdf:resource="&assurance;Type1"/>
  </SymmetricAlgorithm>

  <SymmetricAlgorithm rdf:ID="TripleDES"/>
    <hasNSALevel rdf:resource="&assurance;Type3"/>
  <SymmetricAlgorithm rdf:ID="Blowfish"/>

  <SymmetricAlgorithm rdf:ID="DES">
    <keyLength rdf:datatype="&xsd;int">64</keyLength>
    <hasNSALevel rdf:resource="&assurance;Type3"/>
    <rdfs:comment>Refers only to single DES</rdfs:comment>
  </SymmetricAlgorithm>

  <SymmetricAlgorithm rdf:ID="CAST"/>

  <AsymmetricAlgorithm rdf:ID="RSA"/>
  <AsymmetricAlgorithm rdf:ID="ECC">
    <rdfs:comment>Elliptic Curve Crypto</rdfs:comment>
  </AsymmetricAlgorithm>

  <HashAlgorithm rdf:ID="RIPE-MD"/>
  <HashAlgorithm rdf:ID="SHA-1"/>
  <HashAlgorithm rdf:ID="SHA-256">
    <rdfs:comment>usually used with AES</rdfs:comment>
  </HashAlgorithm>
  <HashAlgorithm rdf:ID="MD5"/>
  <HashAlgorithm rdf:ID="MD4"/>

  <MACAlgorithm rdf:ID="HMAC"/>
  <MACAlgorithm rdf:ID="CBC-MAC"/>

</rdf:RDF>
```

D.4. Security Assurance Ontology (securityAssurance.owl)

```
<?xml version="1.0"?>

<!DOCTYPE uridef[
  <!ENTITY rdf                    "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs            "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY owl                    "http://www.w3.org/2002/07/owl#">
  <!ENTITY xsd                    "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY security          "file://C:\NRLOntologyFiles\securityMain.owl#">
  <!ENTITY assurance         "file://C:\NRLOntologyFiles\securityAssurance.owl#">
  ]>

<rdf:RDF
  xmlns              ="&assurance;"
  xmlns:assurance    ="&assurance;"
  xml:base           ="&assurance;"
  xmlns:rdf          ="&rdf;"
  xmlns:rdfs         ="&rdfs;"
  xmlns:owl          ="&owl;"
  xmlns:xsd          ="&xsd;"
  xmlns:security     ="&security;"
>

 <owl:Ontology>
   <rdfs:comment>
     A security ontology to annotate resources with security-related assurance
information
   </rdfs:comment>
 </owl:Ontology>

 <!-- Top Class; Assurance -->

 <owl:Class rdf:ID="Assurance"/>

 <!--
 Subclasses of Assurance: Standard, Accreditation, Evaluation, Certification
 -->

  <owl:Class rdf:ID="Standard">
    <rdfs:subClassOf rdf:resource="#Assurance"/>
  </owl:Class>

  <owl:Class rdf:ID="Accreditation">
    <rdfs:subClassOf rdf:resource="#Assurance"/>
  </owl:Class>

  <owl:Class rdf:ID="Evaluation">
    <rdfs:subClassOf rdf:resource="#Assurance"/>
  </owl:Class>

  <owl:Class rdf:ID="Certification">
    <rdfs:subClassOf rdf:resource="#Assurance"/>
  </owl:Class>

  <!-- Subclass of Standard class: FIPS -->

  <owl:Class rdf:ID="FIPS">
    <rdfs:subClassOf rdf:resource="#Standard"/>
  </owl:Class>
```

```
<owl:Class rdf:ID="NSA">
  <rdfs:subClassOf rdf:resource="#Standard"/>
</owl:Class>

<!-- Subclass of Evaluation class: TCSEC and CommonCriteria -->

<owl:Class rdf:ID="TCSEC">
  <rdfs:subClassOf rdf:resource="#Evaluation"/>
  <rdfs:comment>Orange Book</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="CommonCriteria">
  <rdfs:subClassOf rdf:resource="#Evaluation"/>
</owl:Class>

<!-- Subclass of Certification class: DITSCAP class -->

<owl:Class rdf:ID="DITSCAP">
  <rdfs:subClassOf rdf:resource="#Certification"/>
</owl:Class>

<!--
Subclass of TCSEC class; DivisionA, DivisionB, DivisionC, and DivisionD
-->

<owl:Class rdf:ID="DivisionA">
  <rdfs:subClassOf rdf:resource="#TCSEC"/>
</owl:Class>

<owl:Class rdf:ID="DivisionB">
  <rdfs:subClassOf rdf:about="#TCSEC"/>
</owl:Class>

<owl:Class rdf:ID="DivisionC">
  <rdfs:subClassOf rdf:about="#TCSEC"/>
</owl:Class>

<owl:Class rdf:ID="DivisionD">
  <rdfs:subClassOf rdf:resource="#TCSEC"/>
</owl:Class>

<!-- Property Declarations -->

<owl:DatatypeProperty rdf:ID="byOrganization">
  <rdfs:domain rdf:resource="#Assurance"/>
  <rdfs:range rdf:resource="&xsd;string"/>
  <rdfs:comment>Specifies which organization is responsible</rdfs:comment>
</owl:DatatypeProperty>

<!-- Instance declarations -->

<!-- FIPS instances -->

<FIPS rdf:ID="FIPS140-2"/>
<FIPS rdf:ID="FIPS180-2"/>
<FIPS rdf:ID="FIPS46-3"/>

<!-- NSA instances -->

<NSA rdf:ID="Type1"/>
<NSA rdf:ID="Type2"/>
<NSA rdf:ID="Type3"/>
<NSA rdfLID="Type4"/>
```

```
  <!-- Division instances -->
  <DivisionA rdf:ID="A1"/>
  <DivisionB rdf:ID="B1"/>
  <DivisionB rdf:ID="B2"/>
  <DivisionB rdf:ID="B3"/>
  <DivisionC rdf:ID="C1"/>
  <DivisionC rdf:ID="C2"/>

  <!-- CommonCriteria instances -->

  <CommonCriteria rdf:ID="EAL1"/>
  <CommonCriteria rdf:ID="EAL2"/>
  <CommonCriteria rdf:ID="EAL3"/>
  <CommonCriteria rdf:ID="EAL4"/>
  <CommonCriteria rdf:ID="EAL5"/>
  <CommonCriteria rdf:ID="EAL6"/>
  <CommonCriteria rdf:ID="EAL7"/>

</rdf:RDF>
```

D.5. Information Object Ontology (infObj.owl)

```
<!DOCTYPE uridef[
  <!ENTITY rdf                  "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs          "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY owl                  "http://www.w3.org/2002/07/owl#">
  <!ENTITY xsd                  "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY algorithms      "file://C:\NRLOntologyFiles\securityAlgorithms.owl#">
  <!ENTITY infObj        "file://C:\NRLOntologyFiles\infObj.owl#">
  ]>

  <rdf:RDF
      xmlns              ="&infObj;"
      xmlns:infObj       ="&infObj;"
      xmlns:base         ="&infObj;"
      xmlns:rdf          ="&rdf;"
      xmlns:rdfs         ="&rdfs;"
      xmlns:owl              ="&owl;"
      xmlns:xsd              ="&xsd;"
      xmlns:algorithms   ="&algorithms;"
   >

  <owl:Ontology>
    <rdfs:comment>
       Ontology to describe security requirements for inputs and outputs of
       services. Security requirements are encrypting, signing of data, etc.
    </rdfs:comment>
  </owl:Ontology>

  <!-- Class declarations -->

  <owl:Class rdf:ID="InformationObject"/>

  <owl:Class rdf:ID="EncInfObj">
    <rdfs:subClassOf rdf:resource="#InformationObject"/>
  </owl:Class>

  <owl:Class rdf:ID="SigInfObj">
    <rdfs:subClassOf rdf:resource="#InformationObject"/>
  </owl:Class>

  <!-- PROPERTIES  -->

  <owl:ObjectProperty rdf:ID="baseObject">
    <rdfs:comment>
      Describes the type or structure of the information
      that is encoded in InformationObject
    </rdfs:comment>
    <rdfs:range rdf:resource="&xsd;Thing"/>
    <rdf:type rdf:resource="&xsd;FunctionalProperty"/>
    <rdfs:domain rdf:resource="#InformationObject"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="cryptoAlgUsed">
    <rdfs:domain rdf:resource="#InformationObject"/>
    <rdfs:range rdf:resource="&algorithms;Algorithm"/>
  </owl:ObjectProperty>

</rdf:RDF>
```

D.6. Service Security Ontology (serviceSecurity.owl)

```
<?xml version="1.0"?>

<!DOCTYPE uridef[
  <!ENTITY rdf                    "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs                "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY owl                    "http://www.w3.org/2002/07/owl#">
  <!ENTITY profile             "http://www.daml.org/services/owl-s/1.1/Profile.owl#">
  <!ENTITY xsd                    "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY serviceSecurity
      "file://C:\NRLOntologyFiles\securitySecurity.owl#">
  <!ENTITY security            "file://C:\NRLOntologyFiles\securityMain.owl#">
  ]>

  <rdf:RDF
    xmlns                      ="&serviceSecurity;"
    xmlns:serviceSecurity      ="&serviceSecurity;"
    xml:base                   ="&serviceSecurity;"
    xmlns:rdf                  ="&rdf;"
    xmlns:rdfs                 ="&rdfs;"
    xmlns:owl                  ="&owl;"
    xmlns:profile              ="&profile;"
    xmlns:xsd                  ="&xsd;"
    xmlns:security             ="&security;"
  >

  <owl:Ontology>
    <rdfs:comment>
      An ontology to annotate OWL-S descriptions with security capabilities
      and security requirements of the resource or service.
    </rdfs:comment>
  </owl:Ontology>

  <!-- Class Definitions -->
  <!--
     Security Concept and Security Objective are made a subclass of
     ServiceParameter, so that we can define capability and requirement as
     a subPropertyOf serviceParameter
  -->

  <owl:Class rdf:about="&security;SecurityConcept">
    <rdfs:subClassOf rdf:resource="&profile;ServiceParameter"/>
  </owl:Class>

  <owl:Class rdf:about="&security;SecurityObjective">
    <rdfs:subClassOf rdf:resource="&profile;ServiceParameter"/>
  </owl:Class>

  <!--
      A ParamValues class that includes both the SecurityConcept class and
      the SecurityObjective class.
      This ParamValue class is used to define the range of values the two
      properties, securityCapability and securityRequirement can possess.
  -->
  <owl:Class rdf:ID="ParamValues">
    <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="&security;SecurityConcept"/>
      <owl:Class rdf:about="&security;SecurityObjective"/>
    </owl:unionOf>
  </owl:Class>
```

```
<!-- Properties -->

<owl:ObjectProperty rdf:ID="securityCapability">
  <rdfs:subPropertyOf rdf:resource="&profile;serviceParameter"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="securityRequirement">
  <rdfs:subPropertyOf rdf:resource="&profile;serviceParameter"/>
</owl:ObjectProperty>

<!-- Property Restrictions -->
<!--
   These restrictions enable the properties to be either a concept or objective -
-->

<owl:Restriction>
  <owl:onProperty rdf:resource="#SecurityCapability"/>
  <owl:allValuesFrom rdf:resource="#ParamValues"/>
</owl:Restriction>

<owl:Restriction>
  <owl:onProperty rdf:resource="#SecurityRequirement"/>
  <owl:allValuesFrom rdf:resource="#ParamValues"/>
</owl:Restriction>

</rdf:RDF>
```

D.7. Agent Security Ontology (agentSecurity.owl)

```
<!DOCTYPE uridef[
  <!ENTITY rdf                "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs              "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY owl                    "http://www.w3.org/2002/07/owl#">
  <!ENTITY xsd                    "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY profile          "http://www.daml.org/services/owl-s/1.1/Profile.owl#">
  <!ENTITY agentSecurity    "file://C:\NRLOntologyFiles\agentSecurity.owl#">
  <!ENTITY security           "file://C:\NRLOntologyFiles\securityMain.owl#">
  ]>

  <rdf:RDF
    xmlns                        ="&agentSecurity;"
    xmlns:agentSecurity ="&agentSecurity;"
    xmlns:base                   ="&agentSecurity;"
    xmlns:rdf           ="&rdf;"
    xmlns:rdfs                   ="&rdfs;"
    xmlns:owl                    ="&owl;"
    xmlns:xsd           ="&xsd;"
    xmlns:profile          ="&profile;"
    xmlns:security      ="&security;"
  >

  <owl:Ontology>
    <rdfs:comment>
      An ontology to enable specification of agent (requestor-side)
      security requirements and capabilities
    </rdfs:comment>
  </owl:Ontology>


  <!-- Class Description -->

  <owl:Class rdf:ID="Agent"/>

  <!--
    A ParamValues class that includes both the SecurityConcept class and
    the SecurityObjective class.
    This ParamValue class is used to define the range of values the two
    properties, securityCapability and securityRequirement can possess.
  -->
  <owl:Class rdf:ID="ParamValues">
    <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="&security;SecurityConcept"/>
      <owl:Class rdf:about="&security;SecurityObjective"/>
    </owl:unionOf>
  </owl:Class>

  <!-- Properties -->

  <owl:ObjectProperty rdf:ID="securityCapability">
    <rdfs:domain rdf:resource="#Agent"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="securityRequirement">
    <rdfs:domain rdf:resource="#Agent"/>
  </owl:ObjectProperty>

  <!-- Property Restrictions -->
  <!-- These restrictions enable the properties to be either a concept or
```

```
    objective -->

    <owl:Restriction>
      <owl:onProperty rdf:resource="#SecurityCapability"/>
      <owl:allValuesFrom rdf:resource="#ParamValues"/>
    </owl:Restriction>

    <owl:Restriction>
      <owl:onProperty rdf:resource="#SecurityRequirement"/>
      <owl:allValuesFrom rdf:resource="#ParamValues"/>
    </owl:Restriction>

  </rdf:RDF>
```